Interface

2003 March

[表紙デザイン：㈱プランニング・ロケッツ]

Contents 2003 March

特集

カードにCPUとOSが載った！

Interface

Contents 2003 March

話題のテクノロジ解説

ショウレポート&コラム

一般解説&連載

情報のページ

連載「やり直しのための信号数学」は,お休みさせていただきます.

インターネットの総合展示会

Internet World Asia 2002

北村俊之

「ITガバナンス向上と，ROIの追求へ」をテーマに「Internet World Asia 2002」が12月4日(水)～6日(金)の3日間，東京ビッグサイトで開催された．主催は(株)IDGジャパン．ブロードバンドの普及とともに，Webは文字や静止画像のみを配信する時代から，動画や音声，音楽を配信する新しいインターフェースの時代へと変化している．こうしたWebの最新の動向をいち早く紹介するのが，本展示会である．また，こうしたブロードバンド時代のネット配信には欠かせないストリーミング技術に関しても「Streaming Media Asia 2002」と題した展示会が併設されていた．最終的な来場者数は3日間で35,258人となった．

● Internet World Asia 2002

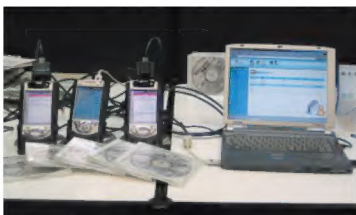
展示会場で，ひときわ大きなブースで来場者の注目を集めていたのが，F5ネットワークスジャパンのブース．こちらではアプリケーショントラフィック管理を実現する「BIG-IP4.5」(**写真1**)を中心に展示を行っていた．こちらの製品はアプリケーションの中味を細部まで読み取れるUIEとiRuleを搭載し，きめの細かいトラフィック管理を実現しているのが大きな特徴であるという．

〔写真1〕**F5ネットワークスジャパンのBIG-IP4.5**

インフォテックでは，Webサイト管理ツール「SiteFlash」およびWeb帳票ツール「Create!Form」によるWebソリューションの展示を行っていた．「Create!Form」は，Web環境においてセキュリティPDFやXMLデータに対応した帳票の出力を可能としている．

マクニカではウィルスチェックおよびコンテンツフィルタの処理を高速化するBlueCoat社の製品(**写真2**)，SSLを利用してVPNを実現するNeoteris社のソリューションのデモなどを行っていた．ジーネットのブースでは，従来のクリック分析とは異なり，マウスの軌跡を計測することによってホームページがどのように閲覧されているかをビジュアルに解析する「ESIシステム」の展示を行っていた．

〔写真2〕**BlueCoatのWeb Security Appliances SG800シリーズ**

マイクロソフトが提供する「Pocket PC 2002」を中心としたビジネスソリューションを一同に展示したPocket PC Pavillionも，来場者の注目を集めていた(**写真3**)．同コーナーでは，東芝，モビマジック，ビービーシステムなど9社がそれぞれのソリューションを展示していた．

〔写真3〕**Pocket PC Pavillionの展示**

ほかには，日本アイ・ビー・エムがWebSphere5ツールの展示を行い，Webサービスに対する同社の取り組みをアピールし，日本ユニシスでは，在庫管理システムや，ポートレットWebサービスによる旅行ポータル，Webサービスを携帯端末から利用できる「MobiThru」などの企業ポータルサービスを多数展示していた．

● Streaming Media Asia 2002

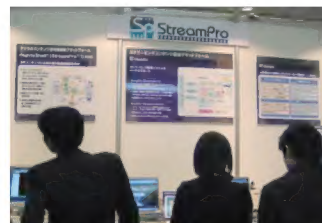
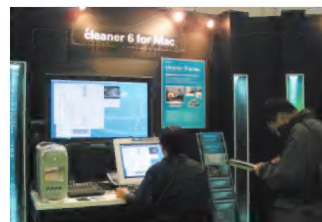
東陽テクニカでは，一台で複数クライアントを接続したのと同じ状況を構築し，ネットワークのパフォーマンス計測を可能にする「WebAvalanche」の紹介を行っていた(**写真4**)．この製品を導入することによって，ネットワークの負荷テストなどにかかるコストが飛躍的に低減するとのことであった．富士ゼロックスではビデオコンテンツの効率的な配信と視聴を実現する「MediaDEPO Server Ver2.0」のデモを中心に展示を行っていた．同製品は，ビデオから代表的なシーンを抽出して一覧にしたり，言葉をキーに音声検索ができるなどの多彩な機能を装備している．

〔写真4〕**東陽テクニカでのWeb Avalancheを用いたパフォーマンス計測の実演**

〔写真5〕**ダイキン工業によるDUONの展示**

ダイキン工業では，ワンソース-マルチユースを実現するための制作・管理ソリューション「DUON」の出展を行い，来場者の関心を集めていた(**写真5**)．日本デジタル・プロセシング・システムズでは，ストリーミング･プリプロセッサ/エンコーダ「StreamZ」の展示を行っており，プリプロセスを専用のハードウェアで行うことにより，画質などの調整をリアルタイムで行えることをアピールしていた．住商エレクトロニクスでは，「Video Solution Server」「MediaBase」「PRISM」といったメディアストリーミングやアーカイブシステムの構築に必要なソフトウェアを多数揃えて展示を行っていた．

ピーエスアイでは，メディアコンバータ，マルチメディア光伝送装置，光ポート付きスイッチングハブなどの展示を行っていた．メディアコンバータは，管理能力が強化されているのが特徴であるとのことであった．

日本電気では，同社のストリーミングソリューションを実際に体験できるコーナーやミニセミナーなどを開催しており，来場者の注目を集めていた(**写真6**)．ディスクリートでは，自動エンコード機能やデュアルCPU対応などの新機能を搭載した「cleaner 6 for Mac」を中心とした展示とデモを行っており，ストリーミング開発者の高い関心を集めていた．同製品は，隣のTooのブースでも取り扱っていた(**写真7**)．

〔写真6〕**日本電気のStreamProの展示**

〔写真7〕**ディスクリートのcleaner 6 for Mac**

Show report

Show & News Digest

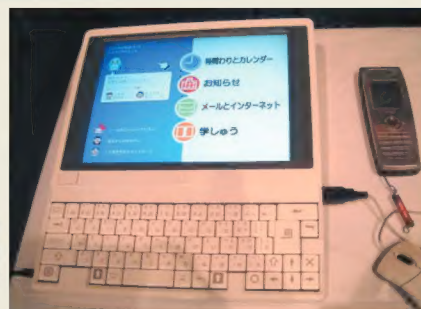
TRONSHOW2003

■日時：2002年12月12日(木)～14日(土)
■場所：ラフォーレミュージアム六本木(東京都港区)

TRONプロジェクトに関する展示会「TRONSHOW2003」が開催された．開催の前日には記者発表会が行われ，坂村健氏よりTRONプロジェクトの成果などが語られた．それによると，昨年発表された組み込み向けTRON開発プラットホームであるT-Engineに関心が集まり，T-Engineフォーラム会員企業が73社まで増加したこと，T-Engineボードが出そろったなどのほか，すでにT-Engineを応用した製品が登場したことも発表された．

また，ユビキタスコンピューティングの関心の高まりとともに，すべての「モノ」に128ビット幅のユニークなID「ユビキタスID」を付加するために，それを管理する「ユビキタスIDセンター」を設立することも発表された．「モノ」は形のあるものだけでなく，ソフトウェアのような形のない「モノ」も対象としていることが特徴である．また，ユビキタスID検索のための通信路はeTRONによりセキュリティが確保される．

TRONプロジェクトの坂村健氏

μT-Engineを元に開発した，三菱電機のIP携帯電話Mobile IP TALK

日立製作所製T-Engineを元に開発した，松下電器の教育用端末ピンチェンジ

Open Source Way

■日時：2002年12月20日(金)
■場所：パシフィコ横浜(神奈川県横浜市)

12月16日～20日の日程で開催されたインターネットの基盤技術に関するカンファレンス「Internet Week 2002」の中で，オープンソースソフトウェアについての理解を深める「Open Source Way」が開催された．

GNUプロジェクトの八田真行氏による「オープンソースとは何か」は，オープンソースの概念や，それにまつわる誤解などに対する解説が行われた．とくにオープンアーキテクチャとの混同，シェアードソースやフリーソフトウェアとの違いなど，オープンソースソフトウェアを扱ううえで重要となる事項が取り上げられた．

そのほかのセミナーは，「オープンソース・ライセンスの実務」「企業戦略としてのオープンソース」「真のオープンソース・ビジネスに必要なエッセンス」など．GPLの無保証条項とPL法における製造者責任の対立，オープンソースソフトウェアをビジネスに結び付ける過程において留意すべき点など，興味深いテーマが扱われた．

セミナーのようす

VON Japan

■日時：2002年12月4日(水)～12月5日(木)
■場所：ヒルトン東京(東京都新宿区)

インターネット電話に代表されるVoIP技術を扱ったカンファレンス＆展示会，VON Japan(Voice on the net Japan)が開催された．カンファレンスは「VoIPの最近の技術動向と日本の取り組み」「VoIP導入戦略～運用と展開～」などが取り上げられた．

併設された展示会では，VoIP構築系のソリューションの展示のほか，KISARAによるSIPプロトコルスタックを使用した「KISARAソフトフォン for PDA」，アルチザネットワークスによるR値，MOS値による音声品質測定が可能なVoIPプロトコルアナライザ「Artiza VoIP Analyzer ver.1.4」などが展示された．

KISARAソフトフォン for PDA

草の根グリッドの心理学

山本　強

IT分野で最近話題になっているキーワードに，「グリッド(Grid)」がある．辞書的な意味は「格子」ということになるのだが，ここで用いられているグリッドの語源は電力分野で使われている送配電ネットワーク，いわゆるpower gridから来ている．

電力送配電のしくみは，じつは情報ネットワークと似ているのである．大きな違いは，流れているのが電力(ワット)であるか，情報(ビット)であるかということである．ネットワークのノード装置が，送配電ネットワークの場合は発電所になり，情報ネットワークの場合はコンピュータになるという違いもある．

グリッドは仮想スーパーコンピュータ

このように，二つのネットワークのモデルは似ているのだが，使われ方はこれまで大きく異なっていた．送配電ネットワークの場合，ネットワーク全体で発電所資源を共用し，見かけ上は一つの巨大発電システムに見えるように構築されている．それに対して，情報ネットワークの場合は任意の2地点間で情報交換することを主たる目的として設計されている．

今回取り上げるIT系のグリッドは，ネットワークの使い方を情報交換だけでなく，計算資源の交換にも使おうというものである．ネットワーク上のコンピュータを高速インターネットで結んで，仮想的な超並列コンピュータを実現することと考えるとわかりやすい．

最近のパソコンの処理速度はGFlopsを超えるのが当たり前になり，ちょっと前のスーパーコンピュータの性能をしのいでいる．これを数百台規模で結合すると，莫大なコンピューティングパワーが手に入ることになる．もっとも，そういう試みは今に始まったわけではなく，CG分野などでは分散レンダリングといって10年以上も前から使われていた手法である．

公的グリッドと草の根グリッド

グリッドの作り方には二つの戦略がある．一つは，ネットワーク上に計画的にコンピュータを配置し，利用権のある人だけが使えるグリッドを構築するものである．多くの場合，国家プロジェクトとして作られるので，ここでは公的グリッドと呼ぶことにしよう．公的グリッドは，予算というエネルギーで動いている．

もう一つは，ブロードバンド経由で接続された個人所有のパソコンを使ったボランティア型のグリッドである．パソコンとネットワークがあれば誰でも参加できるということで，これを草の根グリッドと呼ぶことにする．多くの草の根グリッドは，参加者の善意をエネルギーとして動いている．

草の根グリッドのわかりやすい例に，宇宙の知的生命体を探すSETI@homeプロジェクト(http://setiathome.ssl.berkeley.edu/)がある．これは，電波望遠鏡で宇宙空間からやってくる信号をとらえ，それに含まれているかもしれない知的生命体が発した信号を探索するという計画である．入ってくるデータ量が多いため，それをすべて処理するには莫大な計算パワーが必要になる．昔なら，このようなときはスーパーコンピュータの役目となるのだが，その利用コストとなると半端ではなかった．そこで，ふだんはほとんど眠っている家庭のパソコンに目をつけたというわけである．

草の根グリッドが流行るわけ

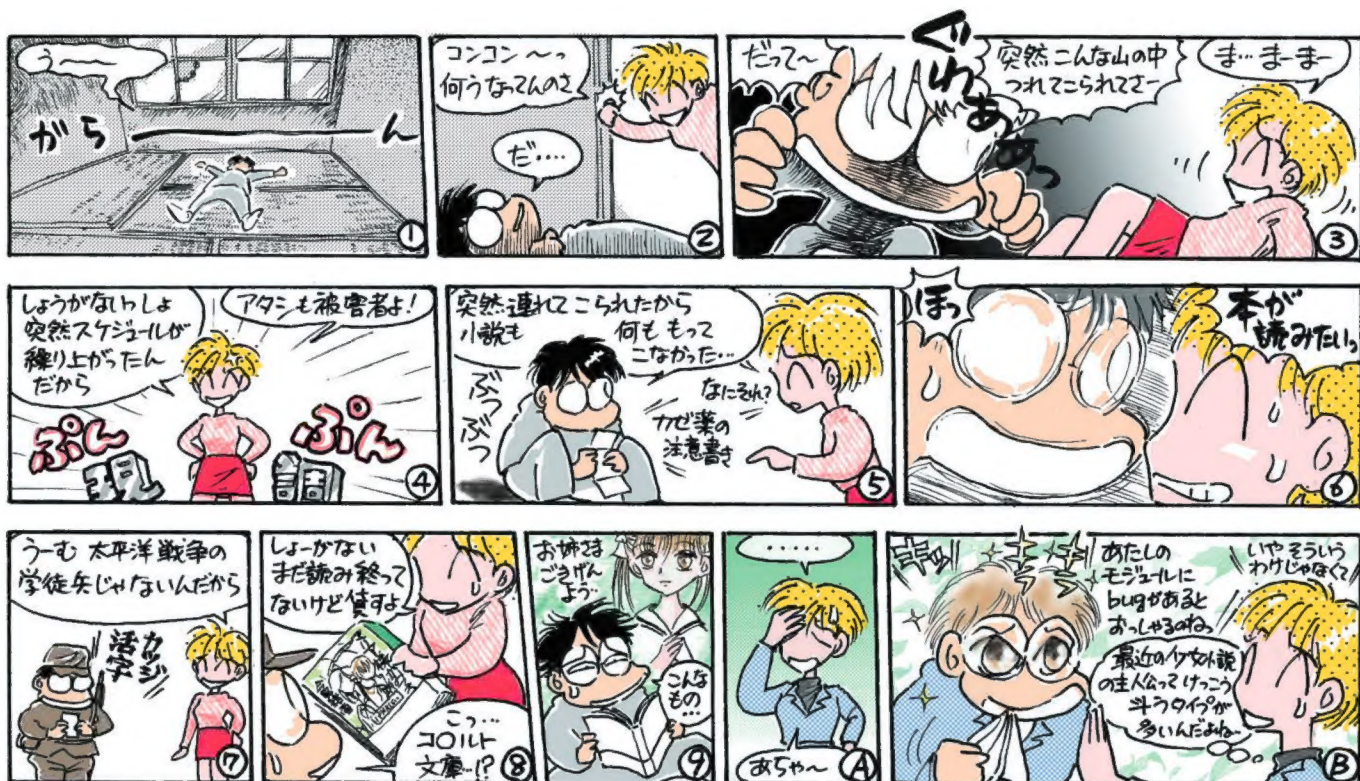
公的グリッドは計画経済型である．必ず利用者がいて，必要な計算パワーを計画配備しているから必要な資源は必ず手に入ることになっている．しかし，草の根グリッドではそうはいかないはずである．それなのになぜ，草の根グリッドに，人はコンピューティングパワーを提供するようになるのだろうか？

じつはもうすでにネットワーク上のコンピュータを結合して実現された仮想巨大情報システムにWWWがある．WWWはグリッドとは違うと思われるかもしれないが，WWWはどう見てもデータグリッドなのである．かつてWWWは誰かに強制されることなく，自然に成長していったのである．どういうわけか，皆が自分のホームページを作りたくなる状況ができたのである．WWWでホームページを公開することのご褒美といえば，自分の仕事や趣味を多くの人に見てもらえること，そしてその評価がアクセス数という形で見えることだった．「豚もおだてりゃ木に登る」というたとえもあるように，人はほめられるとうれしいものである．WWWの黎明期にはそういった種類のご褒美があったように思う．

しかし，SETI@homeのような計算資源提供型のグリッドの場合は，参加者は巨大並列コンピュータの1CPUというなんとも情けない状況になる．それでも多くの人が喜んで計算資源を提供しているのはなぜなのだろうか？

SETI@homeのホームページを見てみると，個人やグループがこのプロジェクトにどのくらい貢献したかのランキング情報が見えるようになっている．いってみれば，見栄を張るための場を用意してあるのである．他の草の根グリッドを見ても，うまくいっているケースは参加者に貢献度競争を促進させるしくみを用意している例が多い．

用語が斬新であるためか，グリッドは画期的なネットワーク分散処理のように見えるのだが，しくみそのものは昔からあったといえる．グリッドの真髄は，個人所有の計算機資源を喜んで提供するような状況を作る心理学にあるのではないかと考えている．

やまもと・つよし
北海道大学大学院工学研究科電子情報工学専攻
計算機情報通信工学講座　超集積計算システム工学分野

フジワラヒロタツの現場検証(**68**)

読書案内(2)

以前，面白がっていただけそうな本を紹介した回が好評いただいたようなので，続きをやりたいなと思っていました．今回は久々にその第2弾をお送りします．

最初に紹介するのは，『ふわふわの泉』(野尻抱介著，ファミ通文庫)です．いわゆるヤングアダルトといわれる世代(中高生ですかね？)向けの文庫で，内容的にもたいがいエンターテイメントしていてさくさく読めます．題材も多岐にわたって，大人が読んでも面白い小説も多く含まれます．楽しく読める小説なのですが，いろいろ深いテーマを平易な文章でわかりやすく料理しています．主人公の眼鏡少女は科学部での実験の最中，偶然から新素材を発見してしまいます．おもしろいのは，そこからベンチャー企業を立ち上げていく過程が，適度なリアリティで描かれている点です．ベンチャーな日常でお疲れの向きは，ぜひ本書で日頃の鬱憤を晴らしてください．

もう一冊，同様に若い人向けであろうハルキ文庫のヌーヴェルＳＦシリーズの『イマジナル・ディスク』(夏緑著，ハルキ文庫)というバイオテクノロジーもののSFを紹介しましょう．著者は理学部の博士課程を終えて小説家になった方のようで，論文と地位の確立に追われる理学部の大学院生の生活が描かれています．どこの分野でも同じように，競争に打ち勝っていくには専門分野に秀でているだけではだめで，権力闘争を泳ぎ切る力が必要なのだなあと，しみじみと読んでおりました．

さて，マンガもぜひ紹介させてください．『BOOM TOWN』(1)～(4)(内田美奈子著，竹書房)です．計算機上に作成された仮想空間「BOOM TOWN」．ここではユーザーが自分の好きな外観で暮らせる，仮想現実の街を提供するサービスをしているという設定のマンガです．その中にはユーザーだけではなくプログラムによって人間そっくりに存在する疑似人格もいたりします．主人公はそこを運営する会社のデバッグ課勤務の多少がさつな女子．エージェントや，自ら作成したソフトウェアツールを駆使して仮想都市のバグを日々デバッグしているという寸法です．

著者はコンピュータを扱っている人種にはいろいろと詳しいらしく，登場人物や，3Dで表現されるユーザーインターフェースがとてもありそうで楽しいのです．アキハバラの怪しげな店が出てきたり，ウィザード君(自称)という冴えない，しかし腕は良さそうな青年がクラッキングを仕掛けてきたり．著者のユーモア感覚は，海野十三[注]を思わせます(古すぎますか……)．掲載誌の都合で，途中で中断しているのが残念です．

こうしてみると，業界を活写している作品が，筆者の琴線にひっかかってくるようです．そういえば，なにかと話題だった暗号冒険小説(？)『クリプトノミコン』(ニール・スティーヴンスン著，ハヤカワ文庫)も，いまひとつと思いつつ，2巻目のはじめの方にある，投資家への目論見書(事業計画書)のカリカチュア(？)に爆笑しました．その筋の方はぜひご一読を．

藤原弘達　(株)JFP　デバイスドライバエンジニア，漫画家

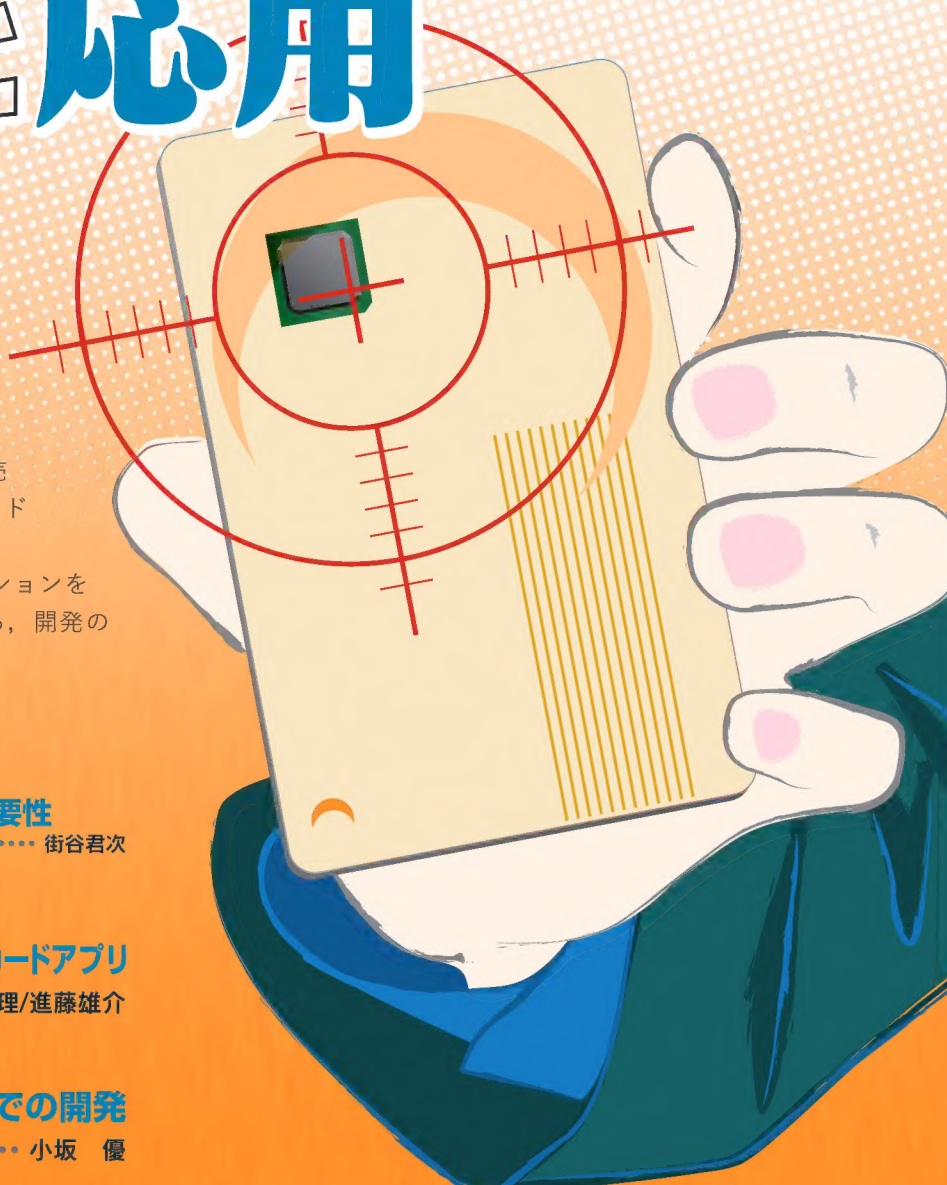
注：海野十三：逓信省電気試験所(後の電総研)研究員．無線，真空管を研究しつつ，科学小説や探偵小説を執筆．本名の佐野昌一名義で技術書もあるらしい．

特集　カードにCPUとOSが載った！

ICカード技術の基礎と応用

これまで使用されてきた磁気カードに代わり，ICカードが注目を集めている．単なる記憶装置にすぎなかった磁気カードと比べ，ICカードはCPUとメモリを搭載し，その中でOSを動作させることにより高いセキュリティを実現している．

ICカード用OSとしてさまざまなものが発売されている．また，非接触カードと接触カードでは使用されるハードウェアが異なる．

そこで本特集では，ICカードアプリケーションを作成するにあたり必要とされる基本知識から，開発の実際，応用例と今後の展望までを解説する．

そしてカードは知性をもった

カード社会とICカードの必要性

街谷君次

あらためていうまでもなく，現代はカード社会である．キャッシュカード，クレジットカード，テレホンカード，そして最近増えてきたポイントカード……と，現代生活にカードは欠かせない存在となっている．

なぜ磁気カードではないのか

これらのカード社会を支える基盤技術として，これまでは磁気カード（磁気ストライプカード）が用いられてきた．これはプラスチックのカードにテープ状の記憶媒体を貼りつけ，これに情報を記録するものだ．テレホンカードをはじめ，さまざまなカードで用いられる磁気カードは，価格の安さ，技術的な実装しやすさなどの要因から，幅広く普及してきた．

しかし近年，磁気カードの存在がおびやかされる問題が出てきた．それはカードの偽造である．磁気カードの課金情報の書き換えなどは，適切な知識と装置さえあれば，比較的簡単に行えるといわれている．また，磁気カードは記憶容量が少なく（数10バイト～100バイト強），あまり多くの情報は記憶できない．また，運用面においても，事実上「1アプリケーション＝1枚のカード」となり，複数のサービスを受けるためには複数のカードが必要になることが多い．あなたも，カードで膨れた財布をお持ちではないだろうか．

なぜメモリカードではないのか

同じように記憶を行うためのカードとしては，各種メモリカードが挙げられる．本誌でもしばしば取り上げられたメモリカードとしては，PCMCIA規格メモリカード，コンパクトフラッシュ，スマートメディア，マルチメディアカードとその上位互換規格であるSDメモリカード，メモリースティック，最近登場した規格としてはxDピクチャーカードなどもある．これらのメモリカードはGバイト級の容量をもつもの，非常に薄くて軽いもの，強固なセキュリティをもち著作権などへの配慮をしたものなど，さまざまな特徴をもっている．

しかし，これらは単純なメモリ＝記憶媒体であり，それ自身がインテリジェントな処理を行えるわけではない．記憶だけならとても素晴らしい媒体ではあるが，それ以上の機能を載せようとすると，外部（カードとの通信を行うデバイス）側で工夫しなければならない．

また，これらメモリカードはPC側の発想から出てきたということもあり，コネクタによる電気的接続が必要になる．ものによっては端子がむき出しになっていることもあり，用途によっては使えない．

そしてICカード

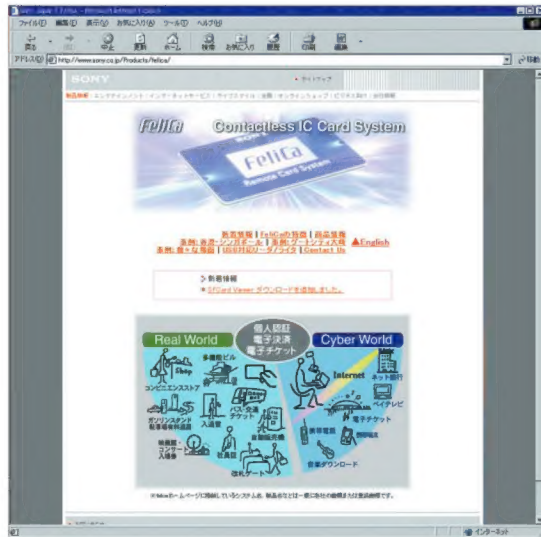
ICカードは，そのような背景から誕生した．ICカードは単純なメモリカードとしてだけではなく（規格上はメモリのみのカード――ワイヤードロジックカードも存在する），その中にCPUを搭載し，OSを動作させている．いわばICカードは，一つの独立したマイコンなのである（**写真1**）．

CPUが入って，プログラム格納用のEEPROMがあり，ワーク用のRAMがあり，データ保存用のフラッシュメモリが載っている……そう考えると，ICカード用アプリケーションの開発は，一般的な組み込み機器アプリケーションの開発と何ら変わりがない．

ICカードもまた，現代の組み込み機器と同様，OSを動作させ，その上でアプリケーションを走らせる．ここで使われるOSは，一般の組み込み機器用OSをさらに小型化した「ICカードOS」である．OSの機能としては，ICカードリーダ/ライタとの通信機能，データ保存を行うためのファイルシステムの管理などがメインとなる．ICカード用OSとしては，本特集で取り上げるMULTOS，ASEPcos，eTRON，FeliCaOSなどが存在し，それぞれが独特の特徴をもっている（**図1**，**図2**）．

さらに意欲的な動きとして，ICカード上にJavaVMを搭載し，Javaでアプリケーションを開発するということも行われている．一般的な組み込み機器もアセンブラやCしか使えなかった時代から，最近ではC++やJavaが使えるなど，開発環境も進化している．Javaを採用したICカードはJavaカードという名で呼

〔写真1〕ICカードはマイコンである！

〔図1〕ソニーのFeliCaのページ
(http://www.sony.co.jp/Products/felica/)

〔図2〕MULTOSの技術仕様を運営管理している，MAOSCOのページ(http://www.multos.com/)

ばれ，本特集ではメールシステムのセキュリティを向上するためのアプリケーションについて解説されている．

ICカードの利点

いうまでもなく，ICカードではプログラムが動作する．すなわち，何でもできるということであるが，現実的にはICカード上で課金情報などを管理するアプリケーションを動作させ，それに対して暗号化処理を行うことなどが多い．単なるメモリカードではそのまま内部のデータを読み書きしていたが，ICカードであれば容易に暗号化などが可能だ（メモリースティックなどはこのあたりに配慮がなされている）．

また，見落としがちな点として，1枚のカードに複数のアプリケーションを搭載することが可能なことが挙げられる．いままで1サービス＝1枚の磁気カードであったものが，複数のサービスを1枚のICカードでまかなうことが可能になる．これで膨れた財布ともおさらばだ．技術的には，複数のアプリケーションの外部からのローディングと実行，ということになる．

ICカードの欠点

ICカードの欠点はとくに見当たらないが，しいて挙げるとすれば価格の高さであろう．カードの実際の価格については各メーカーに問い合わせてほしいが，さすがに磁気カードほどには安くならない．しかし，たとえばSuicaのデポジット（預り金）制度など，新たな疑似（？）購入システムにより，カード発行にかかる費用を軽減するしくみなどが導入されつつある．

おわりに

このようにICカードは，普及段階にある．いままで磁気カードを使用してきた分野をICカードで置き換えるような動きも増えてきた．ちょっとしたポイントカード，社員証，セキュリティカード……このようなシステムを構築する際に，ICカードを選択肢に入れてはどうだろうか．

まちや・きみつぐ　テクニカルライター

ICカードの意外なライバル？

ICカードは手軽に携帯でき，無線による通信ができる．そして，内部でプログラムを実行することが可能で，それによる高いセキュリティを実現することができる．

このように，さまざまな利点が存在するICカードだが，これとほとんど同じことができる機器がすでに存在することはお気づきだろうか？――そう，携帯電話である．

本誌2002年10月号に掲載された「携帯電話の赤外線通信機能を使った入場認証システムの構築」では，IrDA通信機能をもつ携帯電話504iシリーズをもちいて，コンサートなどへの入場管理を行うシステムを構築している．ほかの例ではPDA（携帯情報端末）に赤外線通信モジュールを搭載し，同様のシステムを構築することも可能である．

カードと携帯電話とPDA．このようにまったく性質の異なる機器が，技術の進化とともに同じような機能をもちつつあるということはたいへん興味深い．

それでは，これらの機器は一つに統合されてしまうのだろうか？　筆者はそうは考えない．ICカードで通話をするのは無理があるし，携帯電話をデポジット金500円で配布することは難しい．PDAには電源供給という問題が存在する．それぞれの長所を生かして，これらの機器が独自に進化するのではないだろうか．

Information ― IBMとラショナルソフトウェア，IBMによるラショナル買収で合意
IBMとラショナルソフトウェア・コーポレーションは，IBMが現金約21億ドルでラショナルの株式を買収する最終合意書を両社が締結したことを発表した．

アプリケーションの書き換えが可能な

ICカードOS「MULTOS」によるカードアプリケーションの作成

宇田川真理/進藤雄介

ICカードOS「MULTOS」は，バーチャルマシンの概念を導入し，異なるCPUアーキテクチャでもアプリケーションが動作するという特徴がある．アプリケーションの開発にはアセンブリ言語が使用されるが，C言語やJavaなどのコンパイラを用いることにより，それらの言語での開発も可能になる．また，セキュリティ面でも優れている．

本章では，ICカードで幅広く使用されているOS，MULTOSについて，その概要とプログラミングを含めた実例までを解説する． （編集部）

MULTOSの概要

ICカードは，スマートカードとも呼ばれ，一般にはCPUが搭載されたICが埋め込まれています．これはパソコンにたとえれば，マイクロプロセッサが搭載されているようなものです．その場合，パソコンにさまざまなアプリケーションを追加して使うには，Windowsのようなオペレーティングシステムが必要となります．

MULTOS(MULTi-application Operating System for smart cards)[注1]は，マルチアプリケーションに対応したICカード用オペレーティングシステム(ICカードOS)です．MULTOSの技術仕様は，MAOSCOコンソーシアムによって運営管理されていて，誰でも利用できるオープンな標準仕様になっています．MULTOS，MAOSCOについては，MAOSCOの公式ページ(`http://www.multos.com/`)に掲載されています．

また，MULTOSの普及とその利用環境を向上させるために設立された非営利団体であるマルトス推進協議会のWebページ(`http://www.multos.gr.jp/`)では，MULTOSについて日本語で解説されていますので，一度覗いてみてください．

MULTOSには，以下の特徴があります．

- すでに実用化されている，マルチアプリケーション対応のICカード用オペレーティングシステム
- すでに発行したカードに対して，アプリケーションの追加および削除が可能
- もっとも高いセキュリティレベル(欧州のセキュリティ評価団体ITSECの最高レベルE6)を達成
- 金融や公共ICカードとして幅広く普及している
- 世界各地で幅広い用途のプロジェクトが進行中

それぞれについてもう少し細かく説明しましょう．

注1：MULTOSは，MAOSCOの登録商標．

MULTOSの特徴

● 共通プラットホーム

ICカードやICカード用チップを開発している半導体メーカーはいくつかあります．半導体メーカーによってCPUの違うICチップであっても，MULTOSは共通なバーチャルマシンとAPI(Application Programming Interface)をもつことにより，ICチップの違いを吸収しています．このことによりMULTOSアプリケーションは，異なるCPUをもつMULTOSカードでも共通に利用が可能になります(**図1**)．

また，アプリケーションは，通常ICカードの不揮発性メモリ(EEPROM)上に搭載されるため，搭載アプリケーションをICカードの発行後に追加，または削除できます．さらに，アプリケーション間はファイアウォールで分離されていて，アプリケーションの追加や削除のときにはディジタル証明書が必要なため，高いセキュリティが保持されています．

● 従来型のICカードOSとの相違

MULTOSと従来型のICカードOSとでは，アプリケーションコマンド部の格納場所が大きな相違になります．従来型のICカードOSはほとんどが専用OSであり，アプリケーションを実行する処理コマンドと一緒にROMに書き込まれていて，データ部のみが書き換え可能な不揮発性メモリに格納されます．ROMにアプリケーションのコマンド部が格納されるということは，ICチップ製造時にのみアプリケーションを書き込めるということであり，後からアプリケーションを追加または削除することはできません(**図2**)．

MULTOSの場合は，アプリケーションごとにコマンド部とデータ部が一緒に不揮発性メモリに格納されるので，アプリケーションの書き換えが可能になります．

● MULTOSの実行可能言語

MULTOSの実行可能言語(MEL：MULTOS Executable

〔図1〕MULTOSの特徴

Language)はアセンブリ言語で，MULTOSアプリケーションの開発に利用されます．MELおよびMULTOS-APIは，ICカードでの使用に最適化された構造言語であり，限られたリソース(CPU，メモリなど)を最大限に利用することを焦点に当てて開発されました．アセンブリ言語という特徴から，アプリケーション開発者にとって効率のよいアプリケーション開発が可能になります．さらにC，Java[注2]，VBなどの汎用的な言語でのアプリケーション開発も，MELコードへコンパイルすることによって可能になります(**図3**)．MEL言語仕様は，MAOSCOの公式サイトでアプリケーション開発者ライセンス登録(無料)をすることにより，誰でも入手することができます．

● アプリケーションの追加と削除

MULTOSのアプリケーションの追加/削除は，第三者に悪用されないように，MULTOS鍵管理局(MULTOS Key Management Authority)が発行するデジタル証明書を利用します(**図4**)．MULTOSアプリケーションを追加する際にはロード証明書(ALC：Application Load Certificate)，削除する際にはデリート証明書(ADC：Application Delete Certificate)が必要で，証明書の照合が行われた後に追加/削除が実行されます．

● 通信回線やインターネットによるアプリケーションの追加

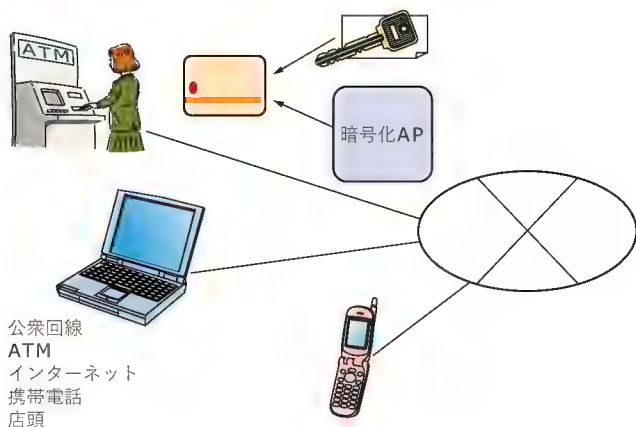
MULTOS上へのアプリケーション追加は，通常の発行環境に限定されず，公衆回線やインターネットに接続されたICカードリーダ/ライタ(端末)を通じて行うことができます(**図5**)．また，アプリケーション追加時にアプリケーションを暗号化することができるので，通信手段そのもののセキュリティへの配慮は最小限にすることができます．また，ロード証明書が必要なので，アプリケーションを追加してもよいMULTOSカードであるかを確実に識別することができます．

〔図2〕従来のOSとの相違

〔図3〕高級言語によるアプリケーション開発

〔図4〕アプリケーションの追加と削除

注2：Javaおよびすべての Java関連の商標およびロゴは，米国およびその他の国における米国Sun Microsystems, Inc.の登録商標．
また，そのほかの会社名および製品名は，それぞれ各社の登録商標または商品名称．

〔図5〕通信を利用したアプリケーションの追加

● 各種暗号鍵や証明書の生成

MULTOSカードは，その製造/発行過程でさまざまな暗号鍵を使い，セキュリティの確保を図っています．

MULTOSカードにアプリケーションを追加して発行する際に使用する暗号鍵と証明書の概略を**図6**に示します．チップメーカーからカードメーカーへのICチップ運搬中の安全を確保するために利用するチップ個別のトランスポート鍵，MULTOSカードを初期化するためにはカード個別秘密鍵や公開鍵を使い，カード発行者が鍵管理局と"会話"するためにはまた別の暗号鍵を使います．

アプリケーションをMULTOSカードに追加するためには，ロード証明書が必要になります．これらMULTOSセキュリティの核となる暗号鍵や証明書は，MULTOS KMAで管理しています．日本を含む全世界に向けサービスを提供する鍵管理局は英国にあり，"Global KMA"と呼ばれ，日本国内専用には(株)日本スマートカードソリューションズ(http://www.jssco.net/)が，日本語で鍵管理局のサービスを提供しています．

MULTOSサプライヤネットワーク

MULTOSはオープンな仕様のため，さまざまな企業がMULTOSカード提供の一役を担っています．半導体メーカー(MULTOS含む)，カード製造会社，アプリケーション提供者がMULTOSカードを発行する会社(イシュアと呼ぶ)をサポートします．

現在，MAOSCOに登録されているおもな会社や製品を**図7**に示します．

MULTOSのロードマップ

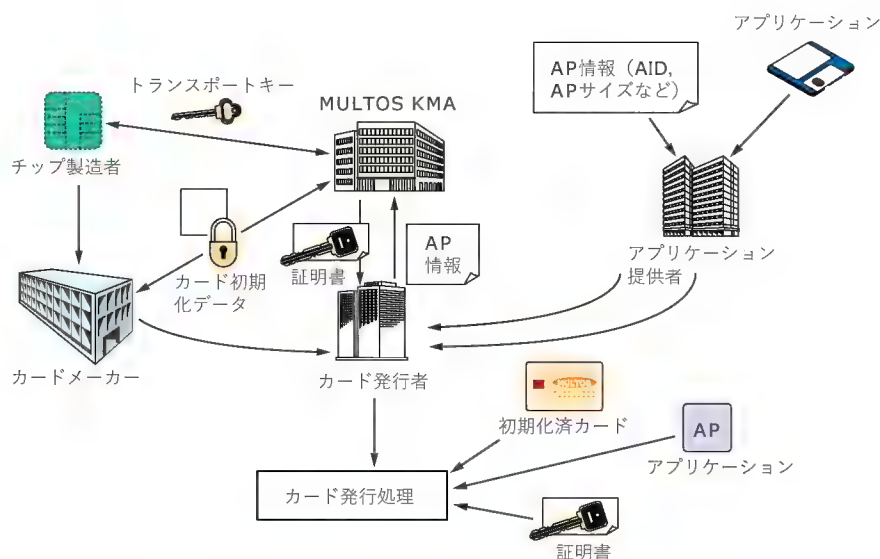
MULTOSは，1997年10月にVersion 3の仕様がリリースされ，EMV V.3.1.1やISO/IEC 7816-3のT=1通信プロトコルサポートのために機能が追加され，現在使用されているものの多くはVersion 4.0になっています．また，搭載されている不揮発性メモリのサイズは8Kバイトから始まり，Version 4では16Kおよび32Kバイトへと変遷してきました．

2000年秋に仕様が確定した次期MULTOSには，変化していく市場ニーズに応えるために非接触方式のICカードのインターフェース機能，新しい公開鍵暗号方式である楕円曲線暗号の機能，さらに第三世代携帯電話の加入者認識モジュールに対応するための機能などが追加されています．

次期MULTOSの新機能

● 非接触ICカード技術への対応

交通関連市場や住民基本台帳カードなどへの参入を目的に，非接触ICカードの仕様がオプションとして追加されました．非

〔図6〕各種鍵と証明書

〔図7〕サプライヤネットワーク

接触ICカードの国際標準規格には，その通信距離に応じて密着型，近接型，近傍型がありますが，MULTOSでは近接型のISO/IEC 14443が採用されました．ISO/IEC 14443をベースにしたMULTOSチップには，以下のような特徴があります．

- 同一チップ上に二つのインターフェース（アプリケーションにより選択）
- ISO/IEC 14443パート1～4対応
- 接触アプリと非接触アプリのデータ交換が可能
- MULTOSのもつ暗号機能（DES，3-DES，RSA，ECCなど）の利用が可能

● 楕円暗号曲線暗号
（ECC：Elliptic Curve Cryptography）技術の導入

楕円曲線暗号（ECC）は，RSA暗号方式に代わる公開鍵暗号方式として注目されている技術で，以下のような特徴があります．

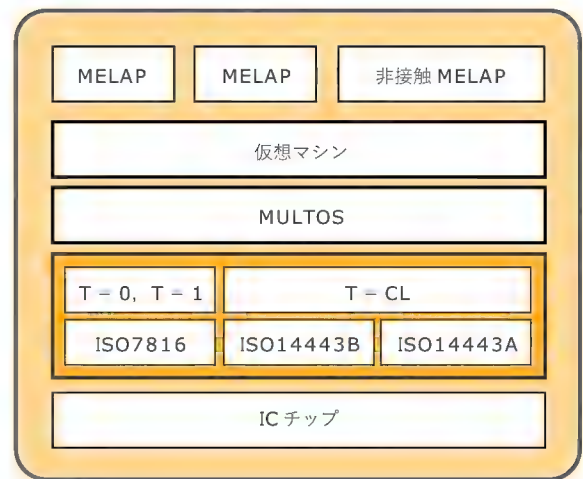
- RSA暗号方式と比較して短い暗号鍵長（RSAの1024ビットはECCの160ビットに相当）で同等の暗号強度をもち，そのため処理速度の向上，メモリの削減，消費電力の減少（非接触モードで有効）が期待できる
- 業界標準（ANSI X9.62，IEEE P1383，ISO 15946）

● その他

非接触対応，楕円曲線暗号対応以外のおもな追加された機能を下記に示します．

- 第三世代携帯電話標準規格検討組織である3GPPがMULTOSに関する二つの規格提案を承認し，MULTOS上のSIM API仕様，APIの評価テスト仕様をリリース予定
- MULTOSカード上での公開鍵/秘密鍵ペアの生成
- ユーザーメモリ（EEPROM）残量通知，ユーザーメモリの再配置（ガベージコレクション），アプリケーションのROMへの搭載（コードレット）など，使い勝手を向上
- Global PlatformとMAOSCOコンソーシアムの連携による，さらなるマルチアプリケーション対応ICカードとしての世界標準化

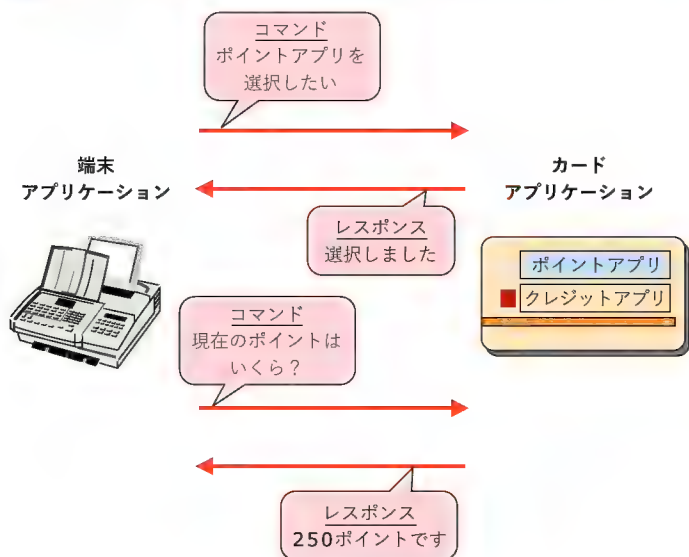
次期MULTOSの構造を**図8**に示します．

〔図8〕次期MULTOSの構造

MULTOSカードを適用したシステム構築

ICカードを動作させて何らかの処理を行わせようと思った場合，ユーザーからの操作を受け付け，カードに対して命令を出す端末アプリケーションと，カードに搭載されている複数のカードアプリケーションが必要になります．

カードアプリケーションは，常に端末から起動されることで処理を開始します．この端末アプリケーションとカードアプリケーションは，端末アプリケーションが送信するコマンドと，これに対してカードアプリケーションが返すレスポンスによって通信しながら処理が進められます．**図9**に示すポイントアプリの例を使用して，これらの処理を簡単に説明します．

端末アプリケーションは，まずカードに搭載されているいくつかのアプリケーションから自分が動作させたいポイントアプリを選択します．これは端末アプリケーションが，まずカード上のアプリケーションを選択するためのコマンド（SELECT FILEコマンドという名称が決められている）をカードに投げることにより行われます．カードアプリケーションは端末からの

〔図9〕ICカードアプリケーションと端末アプリケーション

SELECT FILEコマンドに対して，たしかにそのアプリケーションが存在して選択されたことを回答します．それを受けて，端末アプリケーションは選択したカードアプリケーションに対して「現在のポイント残高はいくらか？」などのカードアプリケーションがもっている利用可能なコマンドを送信し，これに対してカードアプリケーションが適切なレスポンスを返すことで処理を進めていきます．

〔表1〕コマンドの形式

コード	名　前	長さ	説　明	
CLA	クラス	1	命令のクラス コマンドが7816-4にどこまでしたがっているのかなどを表している	
			最上位ニブル[注A1]	意　味
			‘0X’	全産業共通
			‘8X’，‘9X’	コマンドと応答の形式は7816に準拠
			他の値	
INS	命　令	1	命令種別．奇数，‘6X’，‘9X’は使用不可	
P1	パラメータ1	1	コマンド実行の際に渡される補助的なパラメータ．オフセットやモードの指定などに使われる	
P2	パラメータ2	1		
Lc		1 or 3	コマンドデータフィールドに存在するバイト数．第1バイトが0であれば，残りの2バイトで長さを表す	
Data Field		=Lc	コマンド実行時に送信されるコマンドデータ	
Le		1 or 3	レスポンスデータフィールドで期待される最大データバイト長．第1バイトが0であれば，残りの2バイトで長さを表す	

注A.1：CLAのニブルとは，バイト（8ビット）を4ビットごとに分けて論じる必要があるときに用いられる．同一バイト内で，最上位ビット（MSB：Most Significant Byte）から4ビットを上位ニブル，最下位ビット（LSB：Least Significant Byte）から4ビットを下位ニブルという．最上位ビットは，0：業界共通コマンド，8：EMV専用，その他：EMVの範囲外を意味する．また，下位ニブルは，セキュアメッセージングと論理チャネルを表す．たとえばCLA='00'は「業界共通コマンド」であることと，論理チャネル番号が0番であることを表す．

コマンドインターフェース

カードアプリケーションと端末アプリケーションのデータのやりとりは，「コマンド」と「レスポンス」が基本です．このやりとりに関する国際標準が，ISO/IEC7816-part4で決められています．MULTOSカードもISO/IEC7816-part4にしたがっています．よって，設計するコマンドもISO/IEC7816-part4にしたがう必要があります．

ISO/IEC7816-part4では，コマンドはAPDU（Application Protocol Data Unit）と呼ばれる**表1**のような形式を取ることが規定されています．

表1で，CLAからP2までのフィールドはCommand Headerと呼ばれ，必須部分です．残りの部分はCommand Bodyと呼ばれ，必要に応じて出てくるフィールドです．

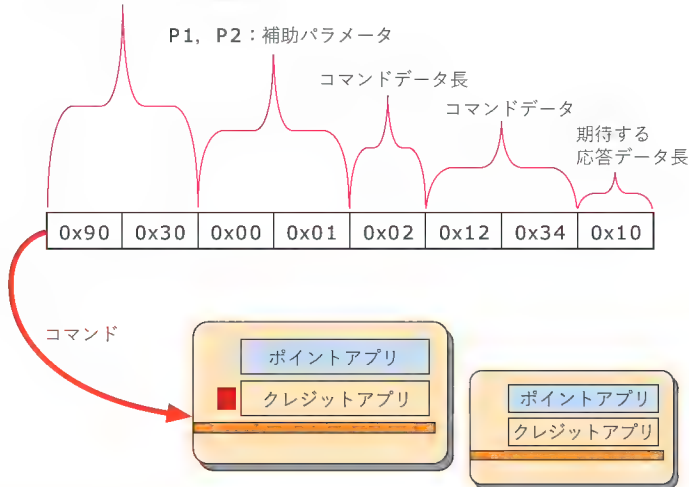
簡単に説明すると，CLAとINSで，送信するコマンドの種類（例：アプリケーション選択，データ取得など）を指定し，Data Fieldにはコマンドの入力パラメータ（例：アプリケーション識別子），Lcにはその長さ，Leにはカードアプリケーションから返ってくるレスポンスデータの長さを指定しています．

このコマンドイメージを簡単にまとめたものが**図10**で，コマンドはこのようなバイナリ列で表せます．

また，レスポンスは**表2**のようなフォーマットで構成されています．

ISO/IEC7816-part4では，カードアプリケーションが返すレスポンスデータと処理結果コードであるStatus wordを区別しています．**表3**に，ISO/IEC7816-part4で規定されているおもなStatus word（以下，SWと略す）を示します．SWの上位バイトをSW1，下位バイトをSW2と表記します．

カードアプリケーションと端末アプリケーションはこれらの定

〔図10〕コマンドのイメージ

〔表2〕コマンドのレスポンス

#	項目名	意　味
1	Data Field	カードアプリケーションが返すレスポンスデータ
2	SW1，SW2	Status word コマンドの処理結果を表すコード

〔表3〕ステータスバイト

	種　別	SW1	SW2	説　明
1	正常処理	'90'	'00'	プロセス完了
2		'61'	'XX'	SW2は応答データのバイト数を示す
3	警告処理	'62'	'XX'	不揮発メモリ状態不変
4		'63'	'XX'	不揮発メモリ状態変更
5	実行エラー	'64'	'00'	不揮発メモリ状態不変
6		'65'	'XX'	不揮発メモリ状態変更
7	チェックエラー	'67'	'00'	長さの誤り
8		'6A'	'XX'	P1，P2のパラメタが誤り
9		'6D'	'00'	命令コード(INS)がサポートされていない，または無効
10		'6E'	'00'	CLAがサポートされていない

義にしたがったコマンドとレスポンスをやりとりすることで処理を行っていくことになります．

MULTOSカードのアーキテクチャ

MULTOSカードアプリケーションの構築方法について記述する前に，MULTOSカードのアーキテクチャについてみることにします[注3]．

● ファイル構成

MULTOSカード内のファイル構成は**図11**のようになります．

各ファイルの詳細はISO/IEC7816に定義されているので，そちらで確認してみてください．

MULTOSのカードアプリケーションは，DF(Dedicated File)として表されます．

カード内の各ファイルには，ファイル識別子(File Identifier)が割り当てられています．とくに，アプリケーションを識別するには，アプリケーション識別子(Application Identifier，AID)を使用します．たとえば，ファイルを選択するSelect Fileコマンドでは，このような識別子を用います．

〔図11〕MULTOSのファイル構成

● データ空間

MULTOSカード内のデータを**図12**に示します．

(1) コード領域

コード領域は，アプリケーションコード用の領域です．この領域は実行するための領域で，書き込んだり読み込んだりすることはできません．

(2) データ領域

データ領域は，アプリケーションが使用するすべてのデータを含みます．この領域は非揮発性領域と揮発性領域からなり，端末や他のアプリケーションのやりとりを行うための領域を含みます．

データ領域は，以下の三つのエリアに分かれます．

● Publicデータ

Publicデータは，端末アプリケーションとカードアプリケーションとの間[注4]でやり取りされるコマンドとレスポンスを扱うための領域です．

● Staticデータ

Staticデータは，カードアプリケーションの不揮発性データを含みます．このデータはアプリケーションのための領域で，端末や他のアプリケーションから直接アクセスすることはできません．

〔図12〕MULTOSカード内のデータ

● Dynamicデータ

Dynamicデータは，カードアプリケーションの揮発性データ領域[注5]です．Dynamicデータは，スタックとセッションデータ

注3：厳密には，カードのアーキテクチャではなく，MULTOSが実現しているAAM(Application Abstract Machine)のアーキテクチャ．
注4：厳密には，カードアプリケーションとリーダライタの間．
注5：Publicデータも揮発性データ．

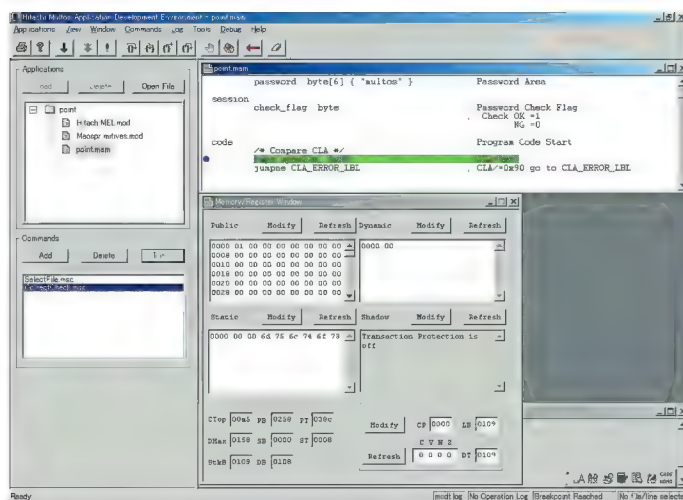
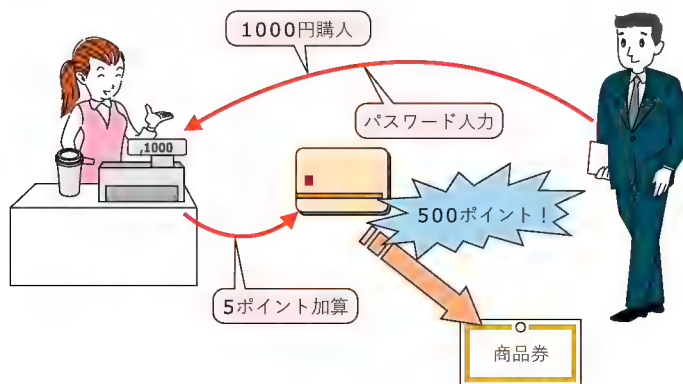
からなります．

スタックは，カードアプリケーションが処理を進めるうえで利用する作業領域です．たとえば，演算結果を一時的に保持したり，Staticデータから読み込んだデータを保持したりします．

セッションデータは，文字どおりセッション中のみ有効なデータです．MULTOSカードにおけるセッションとは，カードに電源供給されている間のあるアプリケーションが選択されてから，次にアプリケーション(選択済みのアプリケーション自身も含む)が選択されるまでの間を指します．セッションデータは，アプリケーションが選択されたとき(セッション開始時に)，Dynamicデータ領域内に0クリアされて領域が取られます．

● レジスタ

MULTOSには，七つのアドレスレジスタ注6と二つの制御レジスタがあります．アドレスレジスタは，StaticデータやPublicデータ，Dynamicデータの位置を示すものです．しかし，後で述べるMALでコーディングする上では，あまり意識することはありません．

〔図13〕アプリケーション構築ツールの画面例

〔図14〕ポイントシステム

カードアプリケーションの作成例

● 開発環境について

MULTOS上のプログラムは，MELと呼ばれるインタプリタ言語により実行されることはすでに述べました．そのため，すべてのMULTOSカードアプリケーションはMELコードに変換する必要があります．日立製作所ではMULTOSカードアプリケーション開発ツールとして，MAL(MULTOS Application Language)と呼ばれるアセンブリ言語で記述したソースファイルをMELに変換するツールと，作成したアプリケーションの動きをシミュレートしデバッグするツール，およびデバッグ済みのアプリケーションをテストカードにロードするツールを用意しています．また，このツールは，実際にカードにロードしたアプリケーションに対してもコマンドを送信しながら動作を確認することのできる機能ももっています．

このツールでは，PC上でICカード上のMULTOS機能をシミュレートしているので，実際のICカードがなくてもPCの閉じた世界でMULTOSアプリケーションを開発・デバッグすることができます．さらにICカードへのプログラムローディング機能を備えているため，MULTOSアプリケーション開発におけるソースコード作成からロードモジュール作成，プログラムテストまですべてのフェーズに対応することが可能です(**図13**)．

● ポイントシステム

ポイントシステムを例に，カードアプリケーションの構築方法を考えてみましょう(**図14**)．このシステムでは，買い物の購入金額に応じて付加されるポイントを保持したカードを，お客がもっています．購入金額の0.5%がポイントとして加算されていき，500ポイントが貯まると500円分の商品券が発行されます．また，カード利用時には，カードの持ち主を確かめるためにパスワードをチェックすることにしましょう．

● コマンドインターフェースの決定

さて，システムがどのような機能をもつのかはわかりました．次は，端末アプリケーションとカードアプリケーションの役割分担です．端末アプリケーションは全体の処理のうち，どの部分を担当するのか？　カードアプリケーションはどうなのか？　ということです．この結果，カードアプリケーションがどのようなコマンドをもつ必要があるのかが決定されます．

MULTOSではカードプログラム内に格納されたデータ(Staticデータ)を操作(読み込み/書き込みなど)するためには，そのデータを操作するためのコマンドを設計し，そのコマンドを扱う処理をカードプログラムでコーディングする必要があります．つまり，コマンドインターフェースは設計者が考える必要があるのです．

たとえば，ポイントの加算処理を考えた場合，次のような二

注6：レジスタは，特定の目的に使用するための，数バイトあるいは数語，ときには数ビットの保持を記憶する機能単位．

つのパターンがあります．

- 端末アプリケーションが現在のポイントを取得し，ポイントの足し算を行い，その結果をカードアプリケーションに渡して，ポイントを更新させる
- 端末アプリケーションは，加算すべきポイント額をカードアプリケーションに渡す．ポイントの加算処理はカードアプリケーション側で行う

前者の立場に立った場合，カードアプリケーションは少なくとも，現在のポイント額を返すコマンドとポイントを指定されたポイント額に更新するコマンドをもつことになります．これに対して後者の場合は，指定されたポイント額を現在のポイントに加算するコマンドをもつことになります．

どちらの立場に立つのかを決定する要因はいろいろあります．なるべく，カードアプリケーションと端末アプリケーションのやりとりを少なくしたいのであれば，後者をとるでしょう．また，なるべくカードアプリケーションのサイズを小さくしたいのであれば，前者をとることになるでしょう．

ここでは，あまり深く考えず**表4**のようなコマンドを用意することにします．

さて，アプリケーションを選択するためのSELECT FILEコマンドはどうなるのでしょうか？ じつはSELECT FILEコマンドはMULTOS OSが応答するコマンドなのです．このようにMULTOS OSが応答するコマンドはほかにもあります．OSが用意するコマンドについては，カードアプリケーションでは実装する必要はありません．

● コマンド設計

それでは，ここまでの知識を生かして，ポイントシステムのカードアプリケーションのコマンドを詳しく設計することにしましょう．

まず，checkコマンドについて考えてみましょう．checkコマンドは，パスワードを送信し，それがカードアプリケーション内で記憶しているパスワードと一致しているかをチェックするコマンドです．したがって，少なくともコマンドデータにチェックすべきパスワードを送信する必要があります．また，コマンドの結果は「一致した」あるいは「一致しなかった」といった情報であり，これはSWで表すことにしましょう．SWの設定値は「一致した」場合はSW=0x9000[注7]，「一致しなかった」場合はSW=0x6300とします．CLAは0x90，INSは0x30，P1・P2は使いません．

同様に，addコマンドはData Fieldに加算ポイントを指定すると，カード内Staticデータとして保存している現在ポイント額に指定された値を足しこむ処理を行います．またgetコマンドは，このコマンドをカードに送信すると，その応答で現在カード内に格納している現在ポイント額の値を応答するような処理を行うこととします．

注7：SW=0x9000は，「SW1=0x90，SW2=0x00」を表すことにする．

〔表4〕作成コマンド一覧

#	コマンド名	機　能
1	check	指定されたパスワードが正しいかチェックする
2	add	checkコマンドが成功していないと実行できない 指定されたポイント額を現行ポイントに加算する 現行ポイントが500ポイントを超えた場合は，現行ポイントを0にし，その旨を通知する
3	get	現行ポイント額を返す

以上，コマンドのインターフェースをまとめると**表5**～**表7**のようになります．

ISO/IEC7816ではコマンドデータの有無とカードアプリケーションのレスポンスデータの有無によりコマンドのケースが定義がされています．

getコマンドのように，コマンドでカードに送信するデータはないが，レスポンスでカードからSW値以外のデータが送られてくるときをケース2と定義しています．またcheckコマンド，addコマンドのように，コマンドでカードに渡すデータはあるが，カードからSW値以外の値が応答されない場合をケース3と呼んでいます．

また，コマンドでデータを送ることもレスポンスでデータを受け取ることもせず，単にカード内処理を行わせる場合をケース1，反対にコマンドでデータを送っており，さらにカードからの応答にもデータが存在する場合をケース4と呼んでいます．このようなコマンドのケースの分類は，後でカードアプリケーションを記述するのに必要になってきます．

● カードアプリケーションのコーディング

すでに述べたように，MULTOSではプログラムをすべてMELオブジェクトに変換する必要があります．

実際のコーディングはC言語またはMAL言語により記述します．ここではおもにMALを使用したアプリケーション構築例を紹介します．

MALではアセンブリ言語をベースとして，

- 手続きの記述と呼び出し
- ユーザー定義のデータ型
- デリバティブ(Cのプリプロセッサコマンド相当)によるインクルードと条件付きアセンブル
- MODELテンプレートによりニモニックの構文定義を自由に変更可能

といった特徴をもっています．実際には，loadやstore，push，popといった命令を駆使してアプリケーションを記述していくことになります．

(1) データ宣言

まず，カードアプリケーションはプログラム内で使用するデータを宣言しておく必要があります．MALでは，データ宣言を**リスト1**のように記述します．

また，データ宣言の前に，そのデータがStaticデータなのか，Sessionデータなのかを宣言します．たとえば，ポイントシステ

〔表5〕check コマンド

コマンド名		check
概　要		指定されたパスワードがカードアプリケーション内のパスワードと一致するか判定する
ケース		3
コマンド		
	CLA	0x90
	INS	0x30
	P1	0x00
	P2	0x00
	Lc	パスワードの長さ
	Data Field	パスワードデータ
	Le	なし
レスポンス		
	Data Field	なし
	SW	0x9000　パスワード一致
		0x6300　パスワード不一致

〔表6〕add コマンド

コマンド名		add
概要		指定されたポイント額を現行ポイントに加算する
ケース		3
コマンド		
	CLA	0x90
	INS	0x32
	P1	0x00
	P2	0x00
	Lc	2
	Data Field	加算すべきポイント額(2バイト) 例：100ポイント→0x0064
	Le	なし
レスポンス		
	Data Field	なし
	SW	0x9000　正常終了
		0x6301　正常終了(ただし，商品券発行時)
		0x6982　エラー：check コマンドが成功していない(未認証によるアクセス拒否)

〔表7〕get コマンド

コマンド名		get
概要		カードアプリケーション内の現行ポイント額を取得する
ケース		2
コマンド		
	CLA	0x90
	INS	0x34
	P1	0x00
	P2	0x00
	Lc	なし
	Data Field	なし
	Le	0x02
レスポンス		
	Data Field	現行ポイント額(2バイト) 例：100ポイント→0x0064
	SW	0x9000　正常終了
		0x6982　エラー：check コマンドが成功していない(未認証によるアクセス拒否)

〔リスト1〕データ宣言

```
label: type
```

〔リスト2〕データ定義例

```
static
    current point: word (0x0000)          ; 現行ポイント額
    password: byte[6] ( "multos" )        ; パスワード
```

ムでは2バイトの現行ポイントデータと6バイトのパスワードデータをStaticデータとしてもつので，**リスト2**のように記述します．byteとwordは，MALがサポートする基本データ型で，それぞれ8ビットと16ビットデータを表します．passwordはbyteの配列型として定義しています．また，current_pointとpasswordの初期値をそれぞれ，0と文字列"multos"にしています[注8]．

(2) コマンド種別の判定

端末アプリケーションから送信されてくるコマンドには，CLAフィールドとINSフィールドがあります．まず，カードアプリケーションは，受信したコマンドのCLAとINSをチェックして，どのコマンドを受信したのかを判断します．その後，各コマンドの処理へ分岐します．

MALでは，1バイトデータの比較は"CMPB"命令で行います．同様に，wordデータはCMPW命令で比較します．さらに，CMP命令に続いて，JUMP命令を用いることで，(条件)分岐を実行することができます．

では，ポイントシステムでのコーディング例を見てみましょう．ポイントシステムで扱うコマンドは，check，add，getの三つです(**リスト3**)．

このコーディングで注目すべき点は，CLAやINSが不正な場合，つまり知らないコマンドを受信した場合の処理です．

このカードアプリケーションは，CLA=0x90，INS=0x30，0x32，0x34のコマンドしか処理しません．つまり，これ以外のCLAやINSをもつコマンドを受信した場合は，エラーとして処理を終了しなくてはなりません．そこで，exitstat命令を使い，知らないCLAの場合はSWに0x6e00を，INSの場合は0x6d00を設定し[注9]，処理を終了します．

また，SWの設定や処理の終了は，setstat命令やexit命令を使って記述することもできます．

なお，明示的にSWを設定せずに処理を終了した場合は，デフォルト値として0x9000(正常終了)が端末アプリケーションに返されます．

注8：このポイントシステムでは，パスワードは変更できないものとしている．

注9：このような場合のSWは，ISO/IEC7816-part4で規定されている．ただし，これとは異なる値のSWを返してもカードアプリケーションは動作する．端末アプリケーションがそれを認識していればよいだけの話．

〔リスト3〕
コマンド分岐のコーディング例

```
code ; プログラムコード開始
    /* CLAのチェック */
    cmpb apduCla, 0x90              ; CLAの値と0x90を比較
    jumpne CLA_ERROR_LBL            ; CLA≠0x90の場合はCLA_ERROR_LBLへ分岐

    /* INSのチェック */
    cmpb apduIns, 0x30              ; INSの値と0x30を比較
    jumpeq CHECK_LBL                ; INS=0x30の場合は, checkコマンドを処理すべく
                                    ; CHECK_LBLへ分岐
    cmpb apduIns, 0x32              ; INSの値と0x32を比較
    jumpeq ADD_LBL                  ; INS=0x32の場合は, addコマンドを処理すべく
                                    ; ADD_LBLへ分岐
    cmpb apduIns, 0x34              ; INSの値と0x34を比較
    jumeq GET_LBL                   ; INS=0x34の場合は, getコマンドを処理すべく
                                    ; GET_LBLへ分岐

    exitstat 0x6d00                 ; INSが0x30, 0x32, 0x34以外の場合は,
                                    ; エラーとし, SW←0x6d00として処理を終了する

CLA_ERROR_LBL:                      ; CLA≠0x90の場合
    exitstat 0x6e00                 ; SW←0x6e00として処理を終了する

/* 各コマンド処理記述 */
...

end ; プログラムコード終了
```

〔リスト4〕
checkコマンドのコーディング例

```
CHECK_LBL:                          ; checkコマンド用ルーチン
    /* ケースをチェックする */
    pushb 0x03                      ; スタックにチェックするケース番号を蓄積
                                    ;(ケース3)
    CheckCase                       ; コマンドのケースをチェック
    jumpne FORMAT_ERROR_LBL         ; コマンドフォーマットが正しくない場合は
                                    ; エラーとする

    /* パスワードの長さチェック */
    cmpw apduLc, 0x0006             ; Lcと0x0006を比較
    jumpne CHK_ERROR_LBL            ; パスワードの長さが一致しない場合は
                                    ; エラー処理へ分岐

    /* パスワードの比較 */
    loadn 6, apduBody               ; チェック用パスワードをスタックに蓄積
    cmpn 6, password                ; カード内のパスワードと比較
    jumpne CHK_ERROR_LBL            ; パスワードが一致しない場合は,
                                    ; エラー処理へ分岐
    setb check_flag, 0x01           ; チェックフラグ←0x01
    exitstat 0x9000                 ; SW←0x9000として処理を終了する

CHK_ERROR_LBL:                      ; パスワードチェックエラー
    setb check_flag, 0x00           ; チェックフラグ←0x00
    exitstat 0x6300                 ; SW←0x6300として処理を終了する
```

(3) 各コマンドごとのコーディング

以上のように，各コマンドごとに処理を分岐したら実際にコマンドが行う処理をMAL言語を使用してコーディングします．各コマンド処理の最初の段階ではそれぞれのコマンドが指定したケースのコマンドフォーマットに従っているかチェックすることが必要ですが，ここでは説明を割愛します．

ケースチェックが正常に行われたと判断した後，実際に処理をコーディングしていきます．コーディングの詳細の説明はここではしませんが，checkコマンドのコーディング例を**リスト4**に掲載しますので参考にしてください．

まとめ

以上，MULTOSアプリケーションの構築について解説しました．ここで記述したのものは本当に簡単な例ですが，プログラミング内容によっては，カード内でも高度な暗号化を行ったり計算をしたりすることが可能になります．

日立製作所ではこれらのアプリケーション構築のサポートだけではなく，ICカード化にあたってのシステム開発のご提案やICカードを扱う端末のご提供，さらに従来のカードからICカード発行への移行など幅広い範囲でのサポートを行っています．これらのサポート内容については日立製作所のホームページに詳細を記述してありますので，

http://www.hitachi.co.jp/Prod/comp/ic-card/main.htm

を参照してください．

うだがわ・まり　(株)日立製作所 ビジネスソリューション事業部
しんどう・ゆうすけ　(株)日立製作所 ビジネスソリューション事業部

高度なセキュリティと拡張性を両立した

ICカードOS「ASEPcos」での開発とセキュリティ

小坂　優

高度なセキュリティを確保するための方式として，公開鍵暗号方式が知られている．これを用いた電子署名などによってデータの改ざんやなりすまし防止などが行われるが，じつはこの方式を採用しただけでは，秘密鍵の露出という問題がある．このように，セキュリティを確保するためには，たんにアルゴリズムだけでなく，運用面においても考慮する必要がある．

そこでICカードOSであるASEPcosを用いて，ICカード内部で電子署名生成を行うなどといった手法がとられる．本章では，ASEPcosを用いた開発とICカードにおけるセキュリティについて解説する．（編集部）

ASEPcosは，（株）アテナ・スマートカード・ソリューションズが独自に開発したICカード用OS（ICカードOS）で，公開鍵暗号を利用したネットワーク・セキュリティ，PKI，電子書名などの利用を目的として設計されています．また，ISO7816標準に加え，日本での利用を念頭に，国内におけるデファクト標準であるJICSAP（ICカード・システム利用促進協議会）仕様2.0版に準拠したシステムアーキテクチャおよびコマンドを採用しています．さらに，コードの95%以上がC言語で記述されており，モジュール構造とあわせて，優れた移植性を備えています．「Pcos」はPortable Card Operating Systemに由来しています．

現在ASEPcosは，当社のASECard Cryptoシリーズに搭載されて出荷されています．ASECard Cryptoシリーズは，RSA公開鍵暗号アルゴリズムを高速に処理するクリプトプロセッサと最大64Kバイトのユーザーメモリを備えるハイエンドICカードですが，性能と同時に，利用しやすさ，開発しやすさも兼ね備えています．

たとえばASECard Crypto（**写真1**）には，アプリケーションの開発を支援する「ASECard Crypto SDK」が用意されています．SDKには，効率的なプログラムの開発を実現するAPIライブラリとファイル設計支援プログラムや，対話型のソースコードジェネレータが含まれます．これらより，プログラマはICカード独特のインターフェースなどに煩わされずに，効率的にアプリケーション開発を行うことができます．

〔写真1〕
ASECard Cryptoの写真

さらにミドルウェアとして，CSP（クリプトサービスプロバイダ）およびPKCS#11ドライバが用意されます．これらを使用することにより，マイクロソフトのCAPI（CryptoAPI）ベースのアプリケーションやPKCSベースのアプリケーションにプラグ＆プレイ感覚で統合することが可能になります．

ICカードOSとは？

ICカードは，メモリだけを備えたシンプルなメモリカードと，マイクロプロセッサを搭載したマイクロプロセッサカードに大別することができます．

マイクロプロセッサカードは，CPU，ROM，不揮発性メモリ（EEPROMなど），暗号プロセッサ，セキュリティ回路を1チップに統合したマイクロプロセッサをベースにしており，ROM内に書き込まれた基本ソフト（ICカードオペレーティングシステム＝ICカードOSまたはカードOS）によって制御されています．

カードOSは，不揮発性メモリ上に階層構造をもったファイルシステムを構築しています．各ディレクトリ，ファイルには個別にセキュリティ条件を設定することができ，このセキュリティ条件を満たした場合にのみ，ディレクトリの移動やファイルへのアクセスが可能になります．これらのセキュリティ管理もカードOSが行っています．

さらに，暗号技術を利用した認証，暗号鍵の生成，電子書名の生成および照合といった暗号技術の活用も，カードOSによって提供されています．このようにカードOSは，まさにICカードの心臓部と考えることができるでしょう．

● PC用OSとの違い

Windowsなどの PC用OSとカードOSとの最大の違いは，

PC用OSがGUIをもち，ユーザーに対して各種のサービスを効果的に提供することを目的としているのに対し，カードOSはデータを安全に格納し，かつ安全に利用することを最大の目的にしており，提供するサービスは限定されているところです．またICカードは，PCに使用されるプロセッサよりもはるかに非力なプロセッサ（今のところ8ビットが主流）と限られたメモリ空間（プログラム領域は100Kバイト以下が普通）で動作する必要があり，PC用OSよりも効率性が求められます．また，リアルタイム性も必要とされます．

PC用OSとのもう一つの大きな違いは，インターフェースです．Windowsに代表されるPC用インターフェースは，GUIによる対話的なインターフェースを備えています．これに対してカードOSは，ISO7816標準で定義されたコマンドAPDU（Application Protocol Data Unit）というフォーマットによる低レベルのインターフェースしか提供していません．

ICカードに送付するコマンドはビット単位での注意深いコーディングが要求されますし，カードからのフィードバックも，コマンドの結果およびコマンドのステータス（成否）を示すエラーコード（2バイト）のみです．このように，今日の一般的なプログラミングと大きく異なる習熟を要求する点が，ICカードの開発を難しくしている要因の一つともいわれています．

● ICカードにおけるセキュリティの必要性

ICカードは，セキュリティアプリケーションと結びつけて語られることの多い技術です．事実ICカードは，ハードウェア，ソフトウェアの相乗効果により，データを安全に格納するメディアとして，きわめて優れた特性を備えています．さらに，ASEPcosに代表される最新のカードOSは，暗号技術の応用面でも優れた特性を備えています．

ICカードチップは，ハードウェア，ソフトウェアの両面でセキュリティが強化された特殊なマイクロコントローラと考えることができます．ハードウェア面では，電気的および光学的な解析手法に対抗するさまざまな対策が施されています．またソフトウェア面では，カードOSのセキュリティシステムによって不正なアクセスから厳重に守られています．

具体的にいうと，個々のデータファイルへのアクセスはカードOSにより厳重に管理されており，データファイルにあらかじめ設定されたセキュリティ条件（暗号キーの認証，照合）が満たされないかぎり，データにアクセスできません．また，セキュリティ条件違反が度重なった場合，不正アクセスが試みられていると判断して，カードをロックさせることも可能です．

一方，電気的，光学的な手法による解析に対しても可能なかぎりの対策が施されており，事実上解析は不可能です．また，解析しようとしてカードからチップを取り出した時点で，チップが破壊されてしまうといった対策も施されています．

このようにICカードは，機密性の高いデータを安全に格納し，携帯するメディアとして，今日もっとも理想的なものの一つであるということができます．

しかし，とくに今日のようなネットワーク環境では，データを安全に格納するだけではセキュリティを十分に保つことができないだけでなく，日々の運用/管理を適切に行わないと，セキュリティホールになる危険性すらあります．そのためカードOSは，データの格納だけでなく，運用においてもセキュリティを強化しています．これは，インターネット環境におけるセキュリティが重要なテーマとなっていることと無縁ではありません．

インターネットの進展は，社会に無類の恩恵を与えました．もはや，インターネットなしに，ビジネスも社会生活も語ることが難しい状態になっています．しかし，インターネットは本質的にセキュリティ問題と表裏一体となっています．

たとえばインターネットは，基本的に匿名性の高いネットワークで，これがある面でインターネットをここまで普及させた要因であるともいえます．しかし，同時になりすましやネット詐欺などの事件を頻繁に引き起こしていることも事実です．これは，とくにビジネスにおけるインターネット利用に対して大きな障害となっています．つまり，インターネットのビジネス利用に際しては，この匿名性というインターネットの本質をいかに克服するか，いかに相手を特定し認証するかということが重要な課題となってきます．これに対するもっとも現実的な解決策の一つが，「デジタル証明書」と「電子署名」です．

● ICカードと公開鍵暗号アルゴリズム

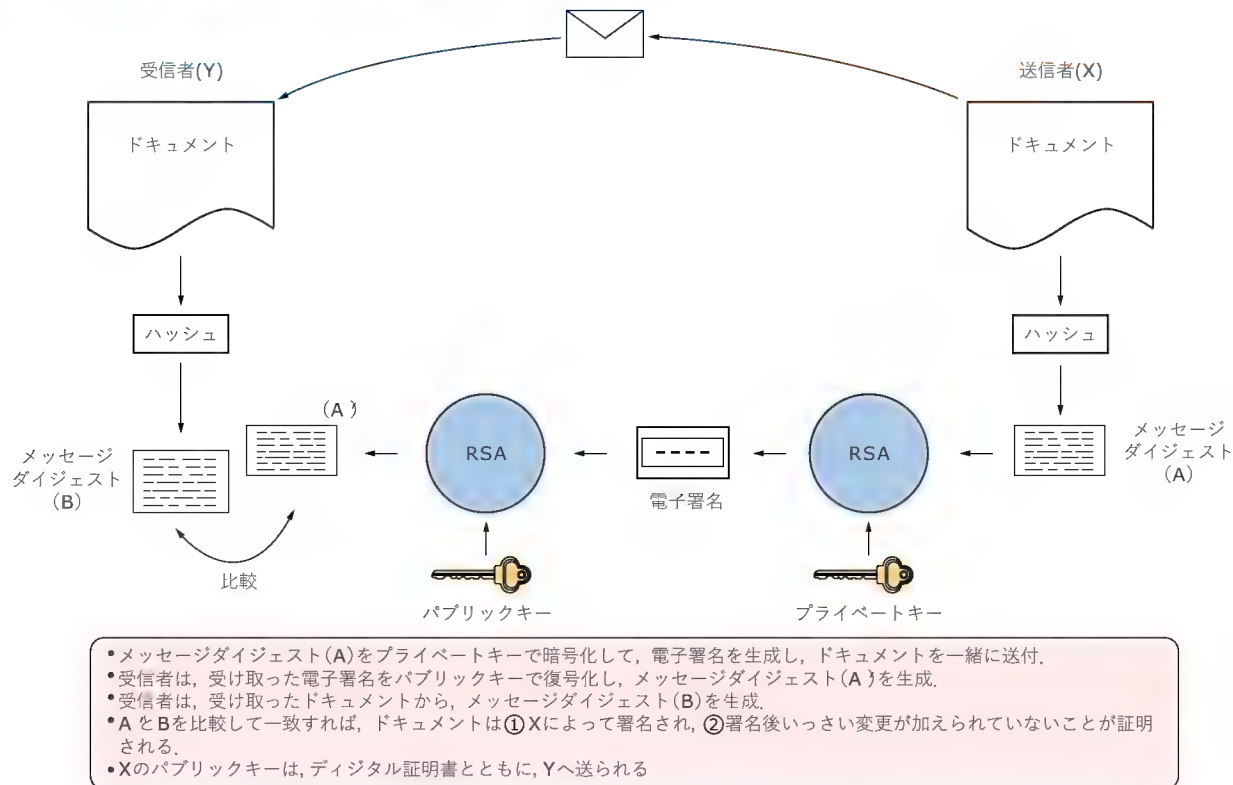
デジタル証明書，電子署名とも公開鍵暗号アルゴリズムという技術をベースにしています．RSAに代表される公開鍵暗号は，データを暗号化する（暗号化）ときと暗号化されたデータを元に戻す（復号化）ときに，別個のキーを使用するという特徴があります．つまり，ある暗号キーで暗号化されたデータは，暗号化の際に使用された暗号キーと対をなす暗号キーでなければ元に戻せないということです．

電子署名は基本的にデータの暗号化で，暗号化の際に使用する暗号キーをプライベートキー，復号化の際に使用する暗号キーをパブリックキーと呼びます．ここで，プライベートキーが安全であるかぎり，それを使って暗号化（電子署名）された電子ドキュメントは，その人のプライベートキーと対をなすパブリックキーでしか復号化することができないため，正しく復号化できれば，その人のプライベートキーで暗号化されたこと，すなわちその人が署名したということが証明されるというのが電子署名の基本的な考えです（**図1**）．また，パブリックキーの信頼性を電子署名の技術を使って保証しているのが，デジタル証明書です．

このプライベートキーを安全に保管/運用するために最適なのが，ASEPcosなどの公開鍵暗号をサポートしたカードOS，あるいはASECard Cryptoのような最新のICカードです．すでに述べたように，ICカードは基本的に機密データ（この場合暗号キー）を安全に保管する機能を備えています．

これだけでは電子署名やデジタル証明書の運用に際して十分なセキュリティを提供することはできません．むしろ，セキュリティホールになる危険性もあります．電子署名を生成する場合，プ

〔図1〕
電子署名

- メッセージダイジェスト(A)をプライベートキーで暗号化して，電子署名を生成し，ドキュメントを一緒に送付．
- 受信者は，受け取った電子署名をパブリックキーで復号化し，メッセージダイジェスト(A)を生成．
- 受信者は，受け取ったドキュメントから，メッセージダイジェスト(B)を生成．
- AとBを比較して一致すれば，ドキュメントは①Xによって署名され，②署名後いっさい変更が加えられていないことが証明される．
- XのパブリックキーはI，ディジタル証明書とともに，Yへ送られる

ライベートキーを使ってRSA公開鍵暗号演算を行う必要があります．旧来のICカードは，このプライベートキーを安全に保管することには長けているものの，RSA演算を行えないので，プライベートキーをいったんPCなどの外部のコンピュータに搬出する必要があります．つまり，プライベートキーを露出してしまうわけです．ここに不正アクセスの危険性が生じます．

ASEPcosおよびASECard Cryptoでは，RSA公開鍵暗号演算をICカードチップ内で完結できるので，プライベートキーをいっさい外部に露出させることがありません．また，RSAキーの生成もICカードチップ内の乱数装置(あるいは外部の乱数装置)が生成した乱数を使って，チップ内で行うことができます．

その際に，チップ内で生成したプライベートキーは，外部に搬出できないように設定することも可能です．こうすることにより，プライベートキーはチップ外に露出される機会がまったくなくなり，インターネットにも対応したきわめて高いレベルのセキュリティを実現できます．

ASEPcosの特徴

ASEPcosは，以下のような目的で開発されたカードOSです．

● PKIまでフルサポート可能な汎用カードOS

ASEPcosは，現時点でICカードに求められるすべての機能を実装することを目標に開発されました．その結果，幅広い用途に利用可能な汎用性を備えています．また，最近需要が急増しているネットワークセキュリティ，PKI(Public Key Infrastructure：公開鍵基盤)の分野に対しても十分な機能とパフォーマンスを備えています．

具体的には，ISO7816-8，9で定義されているコマンド，セキュリティ機能に加え，日本におけるデファクト標準であるJICSAP仕様の最新版である2.0版に準拠しています．

● 最新の暗号システムのサポート

ASEPcosは，共通鍵暗号アルゴリズムとしてDESおよびTriple-DESを，公開鍵暗号アルゴリズムとしてRSAをサポートしています．DESおよびTriple-DESは，暗号化と復号化に同一の暗号キーを使用するアルゴリズムで，高速な処理を特徴としています．

一方，RSAは暗号化と復号化に別々の暗号キーを使用します．そのため，とくにインターネットなど不特定多数が参加するネットワーク環境でのセキュリティに適していますが，処理はより複雑で，高い演算処理能力が要求されます．ASEPcosを搭載するASECard Cryptoシリーズは，RSAを高速に処理する専用のコプロセッサを搭載しており，RSA演算をチップ内で行えます．これにより，RSAプライベートキーをチップ外に搬出せずに，電子署名や電子認証などのRSA演算を行うことができます．

また，チップ内でRSAのキーペア(パブリックキーとプライベートキー)を生成することにより，高レベルのセキュリティを実現できます．また，RSA暗号キー長として最大2048ビットをサポートしています．

● 優れた拡張性

半導体技術の進歩によりICカード用のマイクロコントローラ

も日進月歩で進化しています．ASEPcosは，この技術開発のメリットを生かすため，拡張性を念頭に開発されています．たとえば，ファイルシステムの階層には制限は設けておらず，マイクロコントローラのメモリ容量が許すかぎり階層を設定することができます．現行のASECard Crypto 64Kでは8階層までサポートしていますが，さらにメモリ容量の大きなマイクロコントローラが登場すれば，8階層以上も可能になります．

また，セキュリティ面では，一つのファイルに設定可能なセキュリティキーの数についても，ASEPcos自身では上限を設定していません．これも，ファイルシステムの階層数と同じく，マイクロコントローラのメモリ容量が増えれば，自動的に設定可能なキーの数も増加します．

● 日本におけるデファクト標準のJICSAP仕様への準拠

JICSAP仕様とは，ICカード・システム利用促進協議会が策定したマルチアプリケーション対応のICカード標準で，日本国内におけるデファクト標準として知られています．ASEPcosは，JICSAP仕様の最新版である2.0版(接触)の共通コマンドを100%実装しています．さらに，互換性を実現したうえで，とくにセキュリティ面において独自の拡張を行っています(**表1**)．

● SDK，ミドルウェアによる使いやすさの向上

上でも述べましたが，ICカードの普及が遅れている原因の一つにICカードの独特なインターフェースに起因する，プログラム開発の難しさがあるので，ASEPcosではアプリケーション開発環境やミドルウェアも含めて，トータルな使いやすさを提供することを目標としています．

〔表1〕JICSAP2.0共通コマンドとASEPcos

JICSAP2.0共通コマンド	ASEPcos 実装状況	ASEPcos 独自コマンド
基本コマンド		
SELECT FILE	○	
VERIFY	○	
GET CHALLENGE	○	
EXTERNAL AUTHENTICTATE	○	
INTERNAL AUTHENTICATE	○	
READ BINARY	○	
WRITE BINARY	○	
UPDATE BINARY	○	
READ RECORD(S)	○	
WRITE RECORD(S)	○	
APPEND RECORD	○	
UPDATE RECORD	○	
ERASE RECORD	○	
GET DATA	○	
PUT DATA	○	
管理用コマンド		
CHANGE KEY	○	CREATE DF
DEACTIVATE FILE	○	
ACTIVATE FILE	○	
UNLOCK KEY	○	
発行系コマンド		
CREATE FILE	○	CARD STATUS
DELETE FILE	○	
MANAGE ATTRIBUTE	○	
セキュリティ関連コマンド		
MANAGE SECURITY ENVIRONMENT	○	RSA OPERATION
COMPUTE DIGITAL SIGNATURE	○	DES OPERATION
VERIFY DIGITAL SIGNATURE	○	HASH OPERATION
VERIFY CERTIFICATE	○	
GENERATE PUBLIC KEY PAIR	○	
GET SESSION KEY	○	

ASEPcosを搭載したASECard Cryptoシリーズを利用したICカードアプリケーション開発には，2レベルの開発環境が用意されています．APDUライブラリは，C++，VBといった一般的なプログラミング言語から容易なアクセスを可能にするAPIで，プログラマはICカード独特のインターフェースに煩わされることなく効率的にプログラム開発を行うことが可能です．

CSPおよびPKCS#11ドライバは，より高レベルのインターフェースで，マイクロソフトのCAPIあるいはPKCS仕様に準拠したセキュリティアプリケーションに，プラグ&プレイ感覚でASECard Cryptoを統合することを可能にします．

ASE CardViewは，ASECard Cryptoのファイル設計を支援するプログラムです(**図2**，**図3**)．CardViewにより，Windowsエクスプローラに似た直感的な操作で，ディレクトリ，ファイル，暗号キーを生成し，各ファイルにセキュリティ条件を設定することが可能になります．

● さまざまなICカードチップへの移植性

ASEPcosは，アーキテクチャの異なるマイクロプロセッサに短期間で移植できるように設計されています．具体的には，コードの95%以上をC言語で記述し(**図4**，p.61)，高度なモジュール構造を採用しています．さらに，ハードウェアの違いを吸収するための抽象化レイヤをもつことで，きわめて高度な移植性を実現しています．

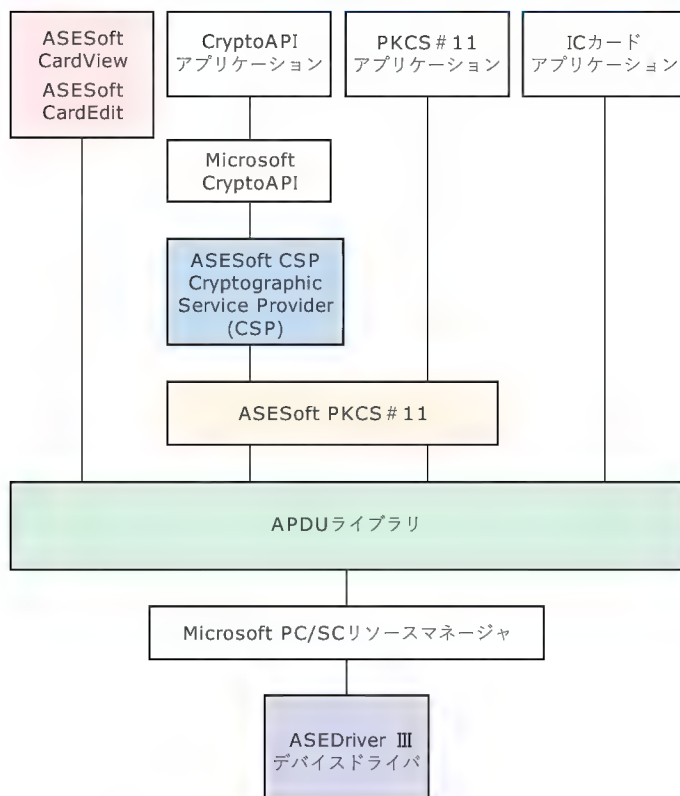
● 効率的なメモリ管理

ICカードは，リソースのかぎられたマイクロコントローラを使用しています．したがって，リソースの有効活用はカードOSを開発する上で常に重要な課題となります．

ASEPcosでは，とくにファイルシステムが利用する不揮発性メモリ(EEPROMが一般的)の管理に独自の技術を投入しています．たとえば，ファイルを削除した際に，そのファイルが占有していた領域はただちに100%開放されます．今日のICカードは，デジタル証明書のような大きなファイルの保管にも用いられるので，メモリスペースのフラグメンテーションの解消は非常に重要な機能の一つです．

● ASEPcosのそのほかの特徴

そのほかのASEPcosの特徴について列挙します．

〔図2〕SDKダイヤグラム

▶一般的な特徴

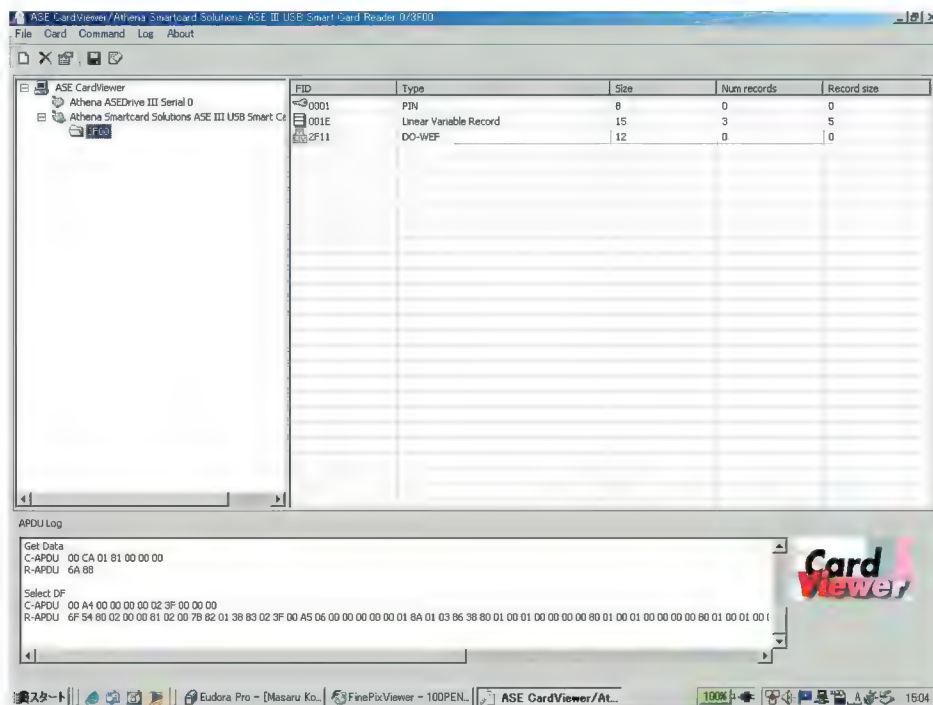
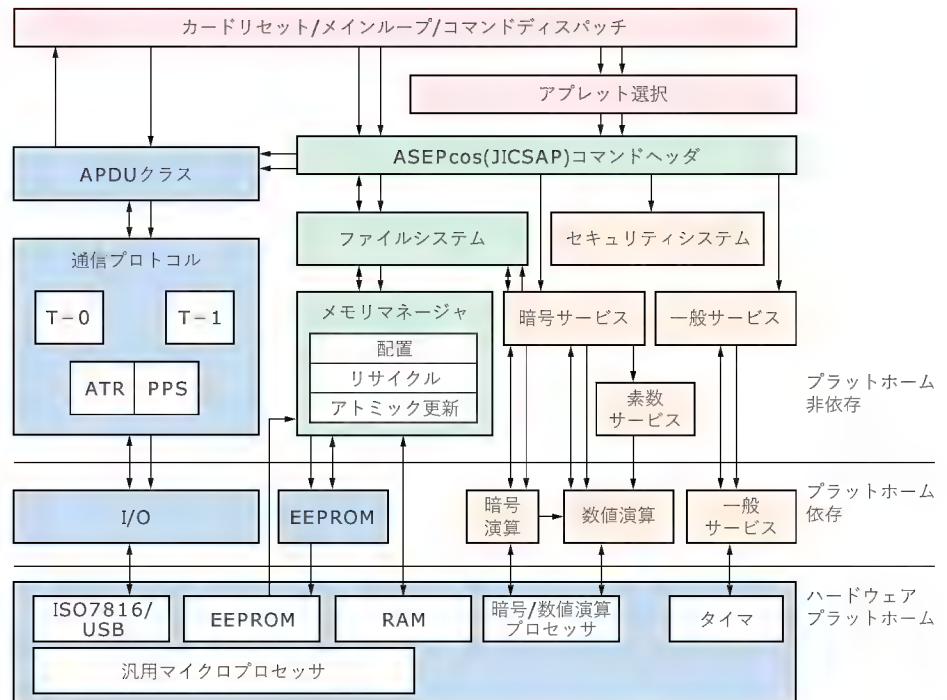
- ISO7816 T=1通信プロトコル(T=0オプション)
- DFごとに複数の外部認証キー設定可能(無制限，ただしASECard Crypto 64Kでは最大16個)
- 階層構造をもつファイルシステム(階層レベル無制限，ただしASECard Crypto 64Kでは最大8階層)
- JICSAP仕様における拡張Le/Lcサポート
- 広範なファイル形式をサポート
  ――透過ファイル，TLVレコードファイル(固定長リニア，可変長リニア，循環)，DO-WEF(単純TLV，BER TLV)
- 優れた移植性
- アプリケーション開発環境の提供
- CSP，PKCS#11などのミドルウェアサポート

▶暗号機能に関する特徴

- RSAサポート(キー長：最大2048ビット)
- DES，Triple-DESサポート(キー長：8，16，24バイト)
- DES，Triple-DES，RSAパブリックキーによる外部認証
- DES，Triple-DES，RSAプライベートキーによる内部認証
- RSAキーペアのカード内生成
- 電子署名のカード内生成，照合
- SHA1ハッシング
- ISO9796-2に準拠した電子署名および電子署名照合
- ISO9796-2に準拠したデジタル証明書からのRSAキーの搬入

ICカードアプリケーション開発の実際

ICカードアプリケーションは，大きく組み込み/リアルタイム系とインタラクティブ系に分類することができます．前者は，自動化されたプロセスの中でICカードを使用する例で，建物への入退出や鉄道バスなどの交通機関の決済，プリペイドカードなどに代表されます．後者はおもにWindowsなどで稼動し，ユー

〔図3〕
マイクロプロセッサカードのファイル設計

〔図4〕ASEPcosブロックダイヤグラム

ザーの介在を前提としたアプリケーションです．ネットワークへのログイン管理や電子署名，セキュアメールなどが代表的です．また，使用するICカードの種類も，アプリケーション開発に大きな影響を与えます．

ICカードは，使用するICチップの種類から，メモリカードとCPUカードに大別することができます(**表2**)．この違いは，ICチップへのアクセス方法，データの取り扱い，セキュリティなど重要な視点が含まれます．また，通信方式から接触型と非接触型に分類することができますが，これらは実のところアプリケーション開発の観点からはあまり大きな違いはありません．

以下にアプリケーションの開発方法について説明し，最後にASE Card Crypto SDKを使用したサンプルを紹介します．

〔表2〕ICカードの分類

	接触/非接触	階層型ファイルシステム	セキュリティ	メモリ容量(Kバイト)	公開鍵暗号
メモリ	接触	×	×	～16	×
CPU	接触	○	○	～64	○
メモリ	非接触	×	×	～1	×
CPU	非接触	○	○	～8	△

アプリケーション開発の流れ

組み込み/リアルタイム系であるか，インタラクティブ系であるかに関わらず，ICカードアプリケーションは，おおむね次のような流れで開発されます．

● システムの設計

システムの設計は，ICカードに限らずすべてのアプリケーションの開発に共通するステップなので，ここではICカードに関わる部分に触れるにとどめます．

まず，ICカードと通信を行うためには，通信プロトコルを正しく実装する必要があります．接触型の場合はISO7816が，非接触の場合はISO14443が中心となります．さらにWindows環境ではPC/SCという標準レイヤが存在します．

PCで稼動するアプリケーションでは，市販のリーダライタを使用するのが一般的です．必要なプロトコルは，すべてリーダライタに実装されています．組み込み系のアプリケーションでは，組み込み用のICカードリーダライタ(ボード)を使用する方法と，マザーボード上にリーダライタの機能を実装する方法があります．後者の場合には，さらにICカードプロトコルを搭載した専用のインターフェースチップを使用する方法と，MCU上に必要なプロトコルを実装する方法があります．

前者の場合，開発者はICカードプロトコルを扱うわずらわしさから開放されますが，MCUを追加することになるためコストアップにつながる可能性があります．後者の場合，すでにあるMCU上にプロトコルを実装するため追加コストは発生しませんが，開発はより複雑にならざるをえません．

● ICカードの選択

ICカードの選択には，大きく二つの基準が存在します．第一はチップの種類で，単純なメモリカードか，より高度なマイクロプロセッサカード(CPUカードともいう)かといった選択です．メモリカードは，数百バイトから数Kバイトのフラットなメモリ空間を提供します．

ファイルやレコードといった概念は存在せず，アプリケーションはメモリセルのアドレスを直接指定して読み書きを行いま

す．メモリカードの一部には，ライトプロテクションを実装しているものもありますが，一般にリードプロテクションは用意されていません．

これに対してマイクロプロセッサカードは，8ビットや32ビットのCPU，メモリ管理機構，不揮発性メモリ（EEPROMが一般的）およびROMを1チップ化した専用のマイコンを使用しています．ICカードの機能は，ROMに記録されたICカードOSによって制御されています．マイクロプロセッサカードでは，階層構造をもったファイルシステムがサポートされており，データはすべてファイルとして管理されます．また，階層の移動やファイルへのアクセスに対しては，セキュリティ条件を自由に設定することができるため，データの不正使用や改ざん，あるいは複数アプリケーションはカードを共有する場合の相互干渉を回避することができます．

マイクロプロセッサカードの発展型として，RSAに代表される公開鍵暗号アルゴリズムを高速に処理するコプロセッサを搭載したカードもあります．これは，公開鍵暗号技術を利用した認証，電子署名など，いわゆるPKIに適したカードとして，おもにネットワーク分野で普及が見込まれています．

メモリカードにするか，マイクロプロセッサカードにするかは，コスト，メモリ容量，セキュリティなどの条件を総合的に判断して決定しなければなりません．実際に，多くのアプリケーションはメモリカードで運用することができますが，セキュリティが必要になる場合，あるいは複数のアプリケーションで1枚のカードを共有する場合にはマイクロプロセッサが不可欠になります．また，データを追記したり頻繁に書き換える場合にもマイクロプロセッサが必要になるケースがあります．

メモリカードはデータを保護する機能がないため，たとえばデータ書き換え中に何らかの理由でカードとアプリケーション間のコネクションが切断された場合，データの整合性は失われます（その時点で書き込みが終了しているセルは新しいデータに，書き込みが終了していないデータは古いデータのままになる）．一部のマイクロプロセッサカードでは，このような場合はデータはいっさい更新されず，書き込み前の状態を保持します．

〔写真2〕接触型ICカードアクセプタと接触型カード

二つ目の基準は通信方式です．通信方式は，接触と非接触に大別されます．接触型は，文字どおり物理的な接点を通じて電源供給と通信を行います（**写真2**）．これに対して非接触方式は，電磁誘導で電力を供給し無線で通信を行います．接触，非接触ともメモリカードとマイクロプロセッサカードが存在します．ただし，非接触は比較的新しい技術であるため，製品展開が接触型に比べて遅れている傾向があります．

たとえば，メモリ容量や公開鍵暗号アルゴリズムなどの点で，非接触型は接触型に比べるとハンディキャップがあります．これに対し，非接触型はカードを実際にリーダに挿入する必要がないため，ユーザーフレンドリーであるということもできます．

このように，接触型と非接触型には一長一短がありますが，おおむね1Kバイト以上のメモリを必要とする場合や公開鍵暗号アルゴリズムが必要とされる場合には接触型を選択するのが現実的です．一方，メモリ容量や高度なセキュリティが要求されない場合には，非接触型も十分現実的な選択肢となります．

● ICカードの設計（ファイルシステム，セキュリティ）

メモリカードの場合には必要ありませんが，マイクロプロセッサカードの場合には，まずファイル構造を設計する必要があります（**図4**）．前述したとおり，マイクロプロセッサカードは階層型のファイルシステムを備えているので，アプリケーションに使用するデータをどのように配置するかを決定し，それぞれにセキュリティ条件を設定します．

セキュリティ条件はファイル，ディレクトリごとに設定することが可能です．セキュリティ条件は，読み出し/書き込みのほか，削除/生成についても許可/非許可を設定することができますし，セキュリティ条件をチェックする方法についても，単純なパスワード的なものから，暗号を利用した認証まで複数の方法が指定可能です．

● プログラムの設計/開発

プログラムの開発には，ICカードに固有の問題はとくにありません．アプリケーションは，ICカードプロトコルを介して，ICカードにコマンドを送り，処理の依頼やデータの送受信を行います．

開発事例

以下では，例としてASEPcosを搭載したASECard CryptoおよびそのSDKとPC/SC標準に準拠したICカードリーダライタASEDrive IIIを使ったアプリケーション開発を紹介します．

● ASEDrive III

ASEDrive IIIは，Windows環境におけるICカード標準であるPC/SCに完全準拠したICカードリーダライタで，ホストインターフェースとしてはUSBとRS-232Cをサポートしています

〔写真3〕PC/SC対応ICカードリーダライタASEDrive III

〔写真4〕組み込み用ICカードリーダライタASEDrive ES-II

(**写真3**，**写真4**)．PC/SCに準拠しているため，Windows上でシームレスにサポートされます．

アプリケーションプログラムは，ASEDrive IIIおよびICカードとは，PC/SCのインターフェース(リソースマネージャ)を介して通信を行っていますが，SDKを使用することにより，プログラマはこれらの低レベル通信を気にせずにプログラミングを行うことが可能になります．

● アプリケーションの流れ

●前処理

ICカードとの通信に先立ち，リーダとのコネクションを確立しなければなりません．また，カードの挿入によりアプリケーションが起動するのが一般的なので，カードの挿入を検出するしくみを実装する必要があります．

1) `ASETalkListReader`関数を使用して，接続されているリーダの中から目的のリーダを検索して，コネクションを確立
2) `ASETalkWaitForCardEvent`関数を使用して，カードの挿入を検出
3) `ASETalkOpenReader`関数で，挿入されたカードを活性化し，コネクションを確立

●データ処理

`ASETalkOpenReader`が正常に終了すると，アプリケーションとICカードは通信可能な状態になります．この状態で，ICカードに対して各種のコマンドを送ります．ここで紹介するサンプルプログラムでは，ASECard Cryptoの公開鍵生成機能を利用して，カード内でRSAキーペア(プライベートキーとパブリックキー)を生成し，それらを使って電子署名およびその照合を次のような手順で行います．

1) PINコードの照合：ASECard Cryptoは，初期状態ではPINコード(パスワード)で保護されている．このパスワードが正しく照合されないかぎり，ASECard Cryptoはいかなるコマンドも受け付けない(サンプルではパスワードを“ASECARD+”に設定している)
2) RSAキーペアの生成：`ASECardCreateRSAKeyApdu`関数，および`ASECardGenkeyPairApdu`関数を使用して，カード内でRSAキーペアを生成する．これらの作業は，ASE CardViewプログラムによりGUIを使って行うことも可能
3) RSAキーペアのセキュリティ環境への登録：これにより電子署名にキーペアを使用することが可能になる
4) 署名に使用するプライベートキーを選択
5) ハッシング：`ASECardHashApdu`関数を使って，署名するデータストリング“Athena”をハッシングし，メッセージダイジェストを生成する
6) 署名：`ASECardComputeDigitalSignatureApdu`関数を使用して，メッセージダイジェストを選択したプライベートキーを使って暗号化(電子署名)する
7) 照合：`ASECardVerifyDigitalSignatureApdu`を使って，電子署名を照合する

●後処理

1) `ASETalkCloseReader`関数をコールして，カードとのセッションを終了し，カードを非活性化する

おわりに

本章では，ICカードの比較的新しい応用分野であるネットワーク・セキュリティ(電子署名，ネットワーク認証など)を中心に紹介しました．ただしこれ以外にも，金融分野や交通分野でもICカードは広く利用されていますし，今後も新しい応用分野が続々と開拓されるものと思われます．

いずれにせよ，ICカードはセキュリティと可搬性を兼ね備えたユニークなデバイスであり，既存サービスの高度化やネットワークや最新技術をベースにした新たなサービスの実現に重要なコンポーネントであるといえます．ICカードは，通信方式，機能，価格によって豊富な選択肢があります．また，プログラミング環境も整備されてきているので，この機会にICカードの導入を検討されてはいかがでしょうか．

こさか・まさる　(株)アテナ・スマートカード・ソリューションズ

〔リスト 1〕サンプルプログラム

```
#include <string.h>
#include "ASEPCOS.h"

int main() {
    ASERESULT     res;
    unsigned long    hReader;
    unsigned char    sw1, sw2;
    unsigned char    reply[64];
    unsigned int     replyLen;
    unsigned char    privData[7];
    unsigned char    pubData[7];
    unsigned char    hashedData[20];
    unsigned int     hashLen;
    unsigned char    publicexp[3] = {0x01, 0x00, 0x01};
    unsigned char*   readers;
    unsigned int     listLen;
    int              cardStatus;

    // リーダリストの大きさを取得
    res = ASETalkListReaders(NULL, &listLen);
    readers = (unsigned char*)malloc(listLen);

    // 利用可能なすべての PC/SC リーダのリストを取得
    res = ASETalkListReaders(readers, &listLen);

    // ここでは, ASEDrive III だけが接続されていると仮定します
    res = ASETalkCreateTalker(readers, &hReader);

    // リーダリストを開放
    free(readers);

    // カードの挿入を監視
    do {
        // 3 seconds timeout
        res = ASETalkWaitForCardEvent(hReader, &cardStatus, 3000);
    } while (cardStatus!= CARD IS PRESENT);

    //  カードの挿入を検出したので, コネクションを確立
    res = ASETalkOpenReader(hReader, PROTOCOL T1, 1);

    // PIN コードを照合
    res = ASECardVerifyApdu(hReader,
                        1, LOCAL KEY,
                        (unsigned char*)"ASECARD+", strlen("ASECARD+"),
                        &sw1, &sw2);

    // RSA パブリックキー(#4)ファイル(512 ビット)を創生
    res = ASECardCreateRSAKeyApdu(hReader, PUBLIC RSA KEY,
                             4, 512, 10,
                             KEY SIGNATURE AUTHENTICATION
                              | KEY ISO9796 2, KEY ATTRIB DEFAULT,
                             publicexp, 3,
                             NULL, 0,
                             0, // no SA
                             &sw1, &sw2);
    // RSA プライベートキー(#3)ファイル(512 ビット)を創生
    res = ASECardCreateRSAKeyApdu(hReader, PRIVATE RSA KEY,
                             3, 512, 10,
                             KEY SIGNATURE AUTHENTICATION
                              | KEY ISO9796 2, KEY ATTRIB DEFAULT,
                             publicexp, 3,
                             NULL, 0,
                             0, // no SA
                             &sw1, &sw2);

    // カードの乱数機能を利用して, カード内で RSA キーペアを生成
    res = ASECardGenKeyPairApdu(hReader, GENERATE RANDOM PAIR,
                             3, 4,
                             &sw1, &sw2);
    // 生成されたキーペアの有効性を確認
    res = ASECardGenKeyPairApdu(hReader, VALIDATE PRIVATE KEY,
                             3, 4,
                             &sw1, &sw2);

    // キーペアをセキュリティ環境に登録.
    // これにより電子署名等で利用可能に.

    privData[0] = 0x89; // tag
    privData[1] = 0x01; // length
    privData[2] = 0x00; // value - level=0
    privData[3] = 0x84; // tag
    privData[4] = 0x02; // length
```

〔リスト1〕サンプルプログラム(つづき)

```
    privData[5] = 0x00; // value - fileId
    privData[6] = 0x03; // value

    // プライベートキー(#3)を電子署名用のキーに指定
    res = ASECardManageSecurityEnvApdu(hReader,
                                     COMPUTATION DECRYPTION INT AUTH,
                                     SIGNATURE COMPUTATION VERIFICATION,
                                     privData, sizeof(privData),
                                     &sw1, &sw2);

    pubData[0] = 0x89; // tag
    pubData[1] = 0x01; // length
    pubData[2] = 0x00; // value - level=0
    pubData[3] = 0x83; // tag
    pubData[4] = 0x02; // length
    pubData[5] = 0x00; // value - fileId
    pubData[6] = 0x04; // value

    // パブリックキー(#4)を電子署名照合用のキーに指定
    res = ASECardManageSecurityEnvApdu(hReader,
                                     VERIFICATION ENCRYPTION EXT AUTH,
                                     SIGNATURE COMPUTATION VERIFICATION,
                                     pubData, sizeof(pubData),
                                     &sw1, &sw2);

    // プライベートキー(#3)を選択
    res = ASECardSelectEFApdu(hReader,
                              SELECTION MODE NO FCI,
                              3,
                              &sw1, &sw2,
                              NULL);

    // プライベートキー(#3)を使って，署名するデータ"Athena"をハッシュ
    hashLen = 20;
    memset(hashedData, 0, 20);
    res = ASECardHashApdu(hReader,
                           (unsigned char*)"Athena", strlen("Athena"),
                           &sw1, &sw2,
                           hashedData, &hashLen);

    // ハッシュされたデータをプライベートキー(#3)で暗号化
    replyLen = 64; // data len is identical to key length
    memset(reply, 0, 64);
    res = ASECardComputeDigitalSignatureApdu(hReader,
                                             hashedData, hashLen,
                                             &sw1, &sw2,
                                             reply, &replyLen);

    // 署名をパブリックキー(#4)で照合
    // by presenting the hashed data
    res = ASECardVerifyDigitalSignatureApdu(hReader,
                                            hashedData, hashLen,
                                            reply, replyLen,
                                            &sw1, &sw2);

    res = ASECardDeleteFileApdu(hReader,
                              3, DELETE EF,
                              NULL, 0,
                              &sw1, &sw2);

    res = ASECardDeleteFileApdu(hReader,
                              4, DELETE EF,
                              NULL, 0,
                              &sw1, &sw2);

    // カードとのセッションを終了
    res = ASETalkCloseReader(hReader);

    return 0;
}
```

身近に存在する採用実績の多いICカード

非接触ICカード技術「FeliCa」の概要

松尾隆史

ICカードの用途として，改札機などに代表される，高速性を要求される分野がある．セキュリティの高さも必要とされるほか，非接触で使用することから，1回のトランザクションが終了しないままカードが通信不能になるなど，さまざまなアクシデントにも対応可能な製品が望まれる．

FeliCaはこれらの機能を満たし，すでにJR東日本のICカードシステム"Suica"に採用されるなど，多くの実績をもつ．本章ではFeliCa技術方式について，その機能と応用例について解説する．

（編集部）

現在，さまざまなカードが発行され，日常生活で身近に利用されています．これらのカードの多くは磁気カードですが，今後はICチップが埋め込まれたICカードに置き換えられると考えられています．これは「記憶容量が大きい」，「セキュリティが高い」，「高い処理能力を備えている」といったICカードの利点によるものです．

非接触ICカードは，従来の磁気カードや接触ICカードとは異なり，カードをリーダ/ライタに挿入することなく「かざす」だけでカードの中の情報を送受信できるICカードです．ICカードとしての特徴をもちながら，接触ICカードと比較して，

- かばんや財布に入れたままでも通信が可能なため，操作性が向上する
- 接触部がなく，カード券面すべてに印刷することが可能であり，デザインの自由度が高い
- 形状がカード型だけに制限されることがなく，ICカード機能をキーホルダーや時計などに組み込むこともできる
- リーダ/ライタとの通信時における接触不良や静電気によるICチップの破壊の危険性が少ない
- カードの接点や読み取り装置のヘッドの磨耗がなく，清掃/定期点検などのメンテナンスが軽減される

などの利点があり，交通機関などを中心に採用が進んできています．

ソニーが開発した技術方式は"FeliCa（フェリカ）"と名付けられ，電子乗車券や電子マネーなど，多様な分野で幅広く利用されています（**写真1**）．

FeliCaは，高速なデータ転送速度を実現したFeliCa無線通信インターフェースと，非接触ICカード用アプリケーションに適したFeliCaOSの採用により，とくに処理速度に関して高いパフォーマンスを実現しています．一般的な処理であれば，1回のトランザクションは0.1秒程度で処理が完了します．これは，非接触インターフェースの特徴である「かざす」ユーザーインターフェースを活かすために非常に重要な要素であると考えています．

FeliCa無線通信インターフェースとFeliCaOS

FeliCaは，非接触ICカードとして，とくに交通用途などで要求される高速処理と高い信頼性を共存させることを可能にし，さらに金融用途での使用にも耐えうるセキュリティの高さももち合わせた技術です．

FeliCaカードは，ICチップが接続されたアンテナを，自然環境への配慮として塩ビではなくPET材で作成されたシートで挟み込んだ構造となっています（**図1**）．リーダ/ライタのアンテナより生成される電磁波を受けると，ICチップはコイル形状のアンテナを利用して電磁波から電力を生成するとともに，同時にリーダ/ライタより送信されたカードコマンドを受信/解析/処理し，レスポンスの返信を行います．

FeliCaの非接触ICカードは，大きく分けて"FeliCa無線通信インターフェース"と"FeliCaOS"との二つの技術要素で構成されています．FeliCa無線通信インターフェースは，無線通信制御部分を指し，FeliCaOSはコマンド体系やファイルシステムを指します．

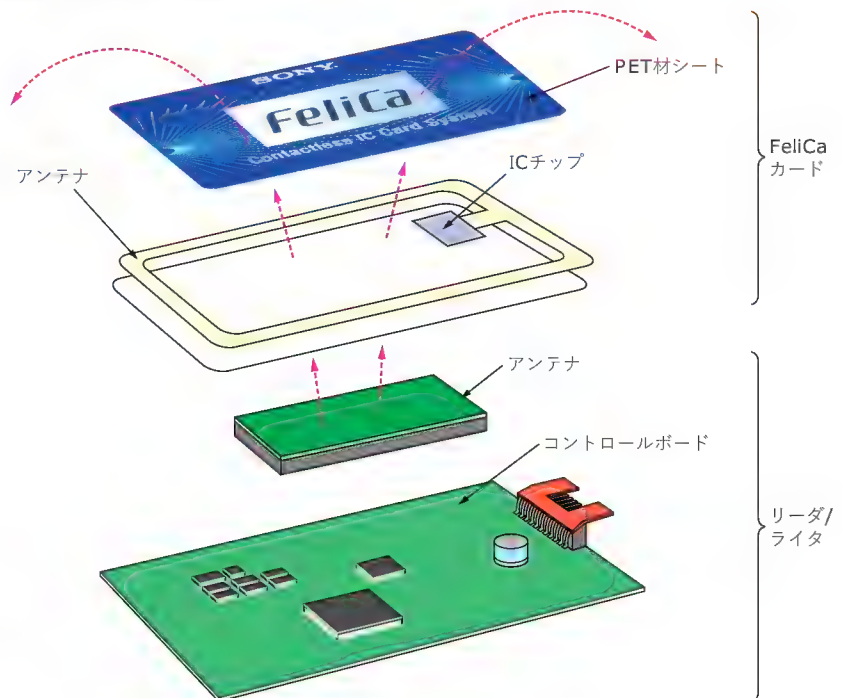
非接触ICカードとしての処理は，FeliCa無線通信インターフェース経由でリーダ/ライタから受信したコマンドをFeliCaOS内でセキュアに処理し，さらにFeliCa無線通信インターフェース経由でレスポンスをリーダ/ライタに返す，ということになります．

FeliCa無線通信インターフェース

非接触ICカードには，その通信距離に応じて複数のタイプが存在します．そのうち，もっとも用途が広いと思われる0～10数cmの通信距離に対応するのが，13.56MHzの搬送周波数を利用する「近接型」です．

近接型は現在，FeliCaのほかにISO/IEC14443として規格化されたTypeA方式とTypeB方式があります．TypeA方式とし

〔写真1〕FeliCaのカードイメージ

〔図1〕FeliCaの内部構造

ては，Philipsが開発したMIFARE仕様が一般的で，比較的早くから実用化されています．また，TypeB方式は，日本においては住民基本台帳カード(IT装備都市カード)などに利用されつつあります．

FeliCaカードは，近接型の非接触ICカードの分類に属しますが，TypeAやTypeBとは異なる独自方式の無線通信インターフェースを採用しています．

このFeliCa無線通信インターフェースでは，FeliCaOSおよびリーダ/ライタ側で生成されたコマンドデータ/レスポンスデータを符号化して送受信を行いますが，その際にはパソコンなどで通常利用される符号化方式とは異なり，Manchester符号化方式と呼ばれる方式を利用して送るべきデータを符号化します．

また，符号化されたデータは，ASK(Amplitude Shift Keying)変調方式という，搬送波の振幅を入力ディジタル信号に対応させて変化させる変調方式で変調され，無線通信が行われます．

● Manchester符号化方式

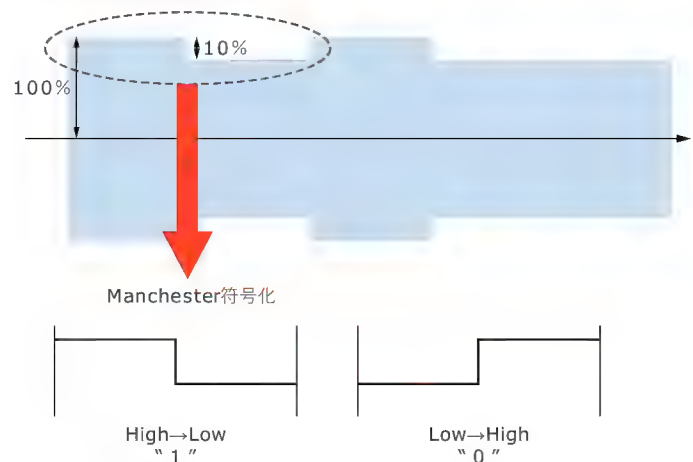
非接触ICカードでは，符号化方式としては主としてNRZ符号化方式やManchester符号化方式が利用されています(**図2**)．

NRZ符号化方式は，データの0と1を単純に信号波形の高低にそのまま対応させるもっともポピュラーな方式で，たとえば，0を「低」，1を「高」として，電圧などの高低でディジタルデータを表現することを可能とする符号化方式です(**図3**)．

FeliCaで利用されているManchester符号化方式はEthernetなどでも利用されており，電圧レベルの変化を利用して符号化を行う方式です．たとえば，ビット区間の中央で電圧レベルを「低」から「高」へ変化させることで「0」を表現し，逆に電圧レベルを「高」から「低」へ変化させることで「1」を表現します．

特徴としては，

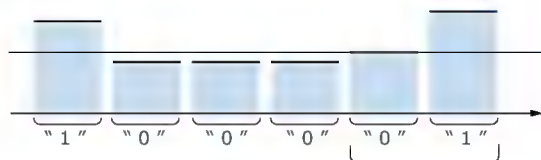
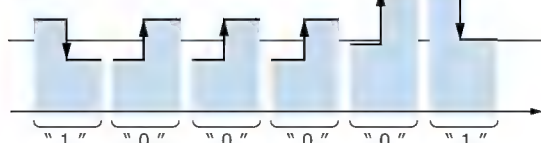
- 電圧レベルが変動しても必ず1ビット内に高低変化があるのでビットを検出しやすい
- 誤り検出能力あり(フルビットで変化がなければ誤り)
- 受信側デバイスは，受け取ったデータストリームから伝送クロックを復元できる(セルフクロッキング方式)

〔図2〕Manchester方式

があげられます．ただし，各ビット区間を二つに分割して情報を伝送するため，変調速度は伝送速度の2倍必要ということになります．

FeliCaでは，

- NRZ符号化方式は，0/1の判定基準を決定することが比較的難しいのに対し，Manchester符号化方式は，0/1の判定基準を決定することが比較的容易であり，判定回路が単純化でき，コストを低廉化できる
- 非接触ICカード利用時の特徴である「通信距離の変動による電圧の変化」に対して耐性が高い(ビットの検知がしやすい)

という理由により，Manchester符号化方式を採用しています．

〔図3〕Manchester 方式と NRZ 符号化方式の比較

〔図4〕ASK変調方式と他の変調方式

● ASK変調方式

搬送波の振幅の変化を利用する変調方式がASK変調方式(**図4**)で，振幅の大小と入力信号を対応させることによりディジタルデータの送受信が可能となります．

ASK変調方式は，構成がシンプルとなるというメリットがある反面，ノイズに弱いという欠点もあります．ただし，非接触ICカードでは，「カードからリーダ/ライタへのレスポンス返送のためカード側でも制御可能な変調方式を利用すべき」という理由により，ASK変調方式が一般に利用されています．

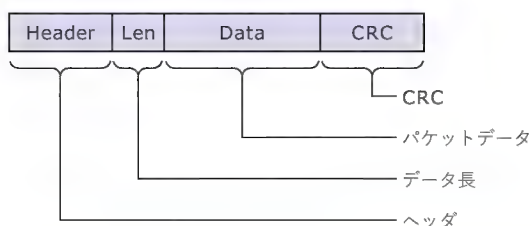
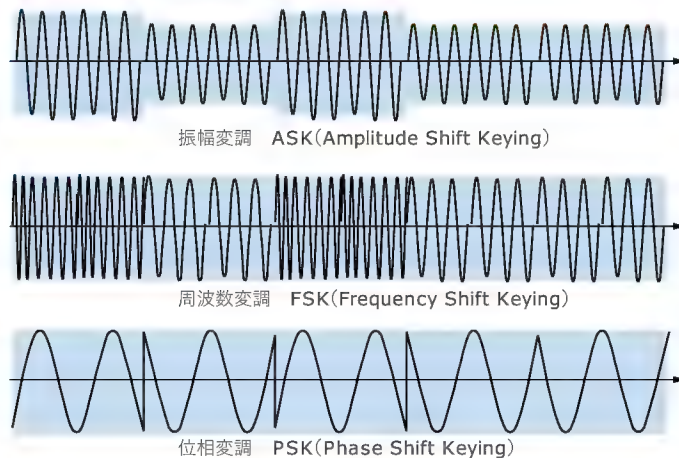
FeliCaでは，とくに振幅の10%程度を変化させる「ASK10%」という方式でディジタルデータの無線通信を可能としています(TypeA方式はASK100%，TypeB方式はASK10%を採用している)．

FeliCaでASK10%を採用した理由としては，

- 電磁波の送出が途切れないため，安定した電源を作りやすく比較的電力の大きいCPUも駆動させることができる
- 副送波を抑えることができるため，電波法の範囲内で，比較的長距離(10数cm程度)の通信距離を確保することが可能

という点があげられます．

なお，非接触ICカード自体は電源をもっていないため，リーダ/ライタからの電磁波による電磁誘導によって電源を発生させ，それによりCPUを動作させています．

〔図5〕FeliCaのパケット構造

リーダ/ライタは，非接触ICカードを動作させるために電磁波を供給すると同時に，送出するコマンドをASK10%を利用して送出します．非接触ICカード側は，受信したコマンドに対して処理を行い，処理結果をレスポンスとしてリーダ/ライタに返送します．その際には，リーダ/ライタからの磁界を受けている状態でカードのICチップ内の負荷を切り替えることにより見かけ上カードから磁界を発生させ，レスポンスを返送しています(ロードスイッチング)．

● パケット構造

FeliCa無線通信インターフェースでは，パケットによりデータをやりとりするしくみを提供しています．

パケットは，**図5**のような構造となっています．パケットデータ部にはFeliCaカードとリーダ/ライタとでやりとりされるコマンドデータおよびレスポンスデータが格納されます．

FeliCaの技術方式について，**表1**にまとめます．

FeliCaOS

● FeliCaOSの特徴

FeliCaOSは，非接触ICカードのために独自に開発されたOSであり，以下のような特徴をもっています．

1) ファイル管理

多階層ファイル構造をもつことができ，各々のファイルに対してアクセス制御情報とセキュリティ鍵を設定することが可能です．

2) トランザクション時のセキュリティ

FeliCaでは，セキュアなトランザクション時にはカードとリーダ/ライタとでTriple-DES暗号アルゴリズムを利用した相互認証を行います．その際に，複数ファイルアクセス時に複数の鍵から一つの「合成鍵」を生成し，合成鍵を用いた相互認証で一括して該当ファイルのアクセスを許可することで，複数ファイルを同時にアクセスする際のトランザクション時間を，セキュ

〔表1〕FeliCaの技術方式

項　目	仕　様
電力伝送	13.56MHz 無変調
データ通信の変調方式（リーダ→カード）	13.56MHz 電力波に対して 10 % ASK
データ通信の変調方式（カード→リーダ）	カード内LSIのロードスイッチング
データ通信の符号化方式（リーダ←→カード）	マンチェスター符号化方式
データ通信速度（リーダ←→カード）	212kbps，424kbps，848kbps
CPU	8ビット RISC CPU
暗号エンジン	ハードウェア DES 処理系
データ通信のエラー検出	CRC
カードICの消費電力	約2～5mW
データ通信の暗号化	相互認証時に発生した乱数をトランザクションごとに変わるトランザクション鍵として用いたブロック暗号
データ通信のシーケンスコントロール	相互認証時に発生した乱数をトランザクションごとに変わるトランザクションIDとして用いたシーケンスコントロール
カード内の実メモリ容量	2Kバイト，4Kバイト，8Kバイト，32Kバイト
カード内のユーザーメモリ容量	1.25Kバイト，2.5Kバイト，5Kバイト，≧25Kバイト
処理中断からのメモリ保護	書き込み専用ブロック（Write Buffer）による保護 8ブロック（16バイト/1ブロック）の同時書き込み保証

リティレベルを落とすことなく大幅に削減しています．

認証後もトランザクションごとに動的に生成する暗号化鍵でデータを暗号化して送受信することで，高いデータ秘匿性を確保しています．

3）マルチアプリケーションとトランザクション異常時のリカバリ

1回のトランザクション中での複数サービスに対する同時処理が可能です．また，ファイルごとのセキュリティ鍵を変えることにより，ほかのアプリケーションからの干渉を防止しています．

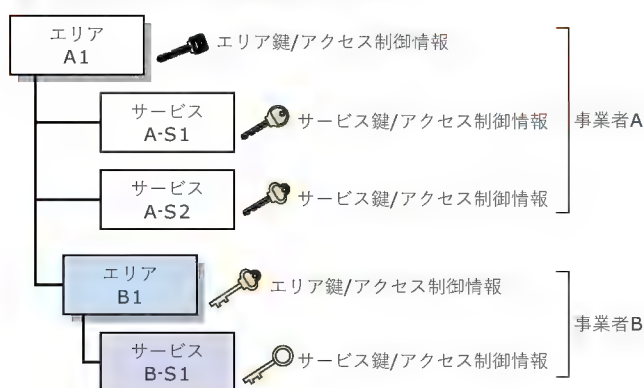
また，データ書き込み処理が正しく完了する前にカードがリーダ/ライタから離れて電力が途切れた場合でも，アクセスするすべてのファイルの整合性を確保します．したがって，上位機器によってリカバリ処理を行う必要がなく，サーバ側もシンプルなシステム構成が可能となります．

● FeliCaOSの機能

▶ファイル管理

● エリアファイルとサービスファイル

FeliCaのファイル構造では，2種類のファイルが存在します．カード内メモリのディレクトリ構造におけるディレクトリを意味するエリアファイルと，具体的なデータファイルを意味するサービスファイルです．

エリアファイルは，通常のパソコンなどで利用しているディレクトリ（もしくはフォルダ）と似た概念です．各エリアファイル内には，複数のサービスファイルを置くことが可能です．下位階層に，さらにエリアファイルを定義することもできます．FeliCaでは，階層の深さは8階層までもつことが可能です．

〔図6〕複数のアクセス権限を設定した例

サービスファイルは，データ本体をカードに蓄積するために利用します．FeliCaのメモリは，16バイト単位のブロックとして区切られており，各々のサービスファイルは，複数個のブロックにより構成されます．

サービスファイルには，大きく分けて三つの種類が存在します．サービス提供者は，これらのファイル種別を組み合わせ，自分の提供するアプリケーションに応じて柔軟にサービスファイルの構成を選択することができます．

1）ランダムアクセスファイル

もっとも一般的なファイルで，データを指定した場所にそのまま読み書きできます．

2）サイクリックアクセスファイル

新しいデータを一つ追加すると，もっとも古いデータが一つ消えるしくみをもちます．最新のログを保存する場合に使用します．

3）パースアクセスファイル

すでにある数字（お金情報など）から減算する機能をもちます．また，一度減算した数字を再度加算する機能などもあります．

● ファイルの属性

エリアファイルおよびサービスファイルには，それぞれ「読み出し書き込み両用/読み出しのみ」と「セキュリティ鍵なし/あり」のアクセス制御情報を設定することができます．これにより，各ファイルのアクセス権限を細かく設定することができます．

また，一つのサービスファイルに対して複数のアクセス権限の，異なる属性を付けることが可能です（**図6**）．たとえば，あるサービスファイルに対して，

1）セキュリティ鍵ありでアクセスする場合には，読み出し/書き込みともに可能

2）セキュリティ鍵なしでアクセスする場合には，読み出しのみ可能

という二つのアクセス制御情報を設定することにより，セキュリティ鍵を管理している人のみがサービスファイルに書き込む

ことができ，その他の人はデータを参照することだけができるサービスファイルを作成できます．

●セキュリティ鍵

各エリアファイルおよびサービスファイルには，それぞれエリア鍵およびサービス鍵といったセキュリティ鍵が設定され，セキュリティ鍵がわからないとサービスファイルにアクセスしたり，サービスファイルを追加したりすることができないようなしくみをもっています．これにより，各エリアごとのセキュリティを確保することが可能となります．

これを，**図7**の例を用いて説明します．たとえば，サービス提供者(エリアファイル管理者)1は，サービス提供者2のサービスファイル2-1のセキュリティ鍵がわからない場合，基本的にサービスファイル2-1にはアクセスすることができませんが，サービス提供者1はサービス提供者2の許可を得られれば，サービスファイル2-1のデータをアクセスすることができるようになります．

このとき，FeliCaではサービス提供者1にサービスファイル2-1のセキュリティ鍵自体を知らせずに，セキュリティ鍵から生成した特定のデータをサービス提供者1に教えることにより，サービス提供者1に対してサービスファイル2-1をアクセスさせることができるしくみを提供しています．

この機能により，たとえばサービス提供者1とサービス提供者2が共通カードを発行した場合に，サービス提供者1がサービス提供者2のサービスファイル2-1をアクセスできるようにすることで，サービス提供者1自身のチケットサービスとサービス提供者2の電子マネーサービスを連携させることが可能となります．

〔図7〕セキュリティ鍵を用いた例

▶トランザクション時のセキュリティ

FeliCaのセキュリティは，Triple-DES暗号アルゴリズムによる相互認証を行うとともに，読み取り，書き込み(記録)に関しても暗号化を施すことにより確保しています．

FeliCaでは，ワンタイム乱数を用いた動的な認証を使用しています．また，データそのものに関して，改ざん防止/すり替え防止/暗号化を行っており，高いセキュリティを保っています．

●相互認証

FeliCaの相互認証(鍵がお互いに正しいかを確認する処理)方式は，ISO/IEC9798で規定される3パス相互認証方式を強化したFeliCaオリジナルの相互認証方式を採用することで，処理の高速化を実現しています．

管理者Aがサービスファイル1とサービスファイル2をアクセスする場合，セキュリティ鍵は「ルート管理者エリア」，「管理者Aエリア」，「サービスファイル1」，「サービスファイル2」に別々につけられているために，通常は，四つそれぞれについて，相互認証を行わなければなりません．一般的に，相互認証は通信回数が多いために処理時間がかかり，四つそれぞれに対して相互認証を行った場合には，高速な処理を望むことができません．

そこでFeliCaでは，四つのセキュリティ鍵を一つのセキュリティ鍵に合成し，この鍵を利用して相互認証を一度だけとすることで，処理を高速化しています．このセキュリティ鍵の合成により，ファイル数が増えた場合にも高速な処理が実現されています．この鍵の合成手法は，サービスファイル数を最大8までサポートしています．

●トランザクションIDとトランザクション鍵

相互認証時に，動的に「トランザクションID」および「トランザクション鍵」を生成し，これらを利用して以降の暗号通信を実施しています(**図8**)．

〔図8〕トランザクションIDとトランザクション鍵を使用した例

トランザクションIDは暗号化対象データの先頭に置かれ，ブロック暗号におけるCBCモードを利用することで暗号化データの強度を高めるとともに，トランザクションごとに更新および確認を行うことで，同一データをセッション中に一度だけしか使用できないように制限をかけることが可能です．

また，トランザクション鍵をセッションごとに異なる使い捨ての鍵とすることで，安全性を保つことができます．これにより，たとえ送信データが同じであっても，毎回異なる暗号化バイト列とすることにより，セキュリティレベルを向上させています．

● 改ざん検出バイトの利用

暗号化の前に，特定のアルゴリズムを利用してデータに依存した改ざん検出バイトを計算し，データに付加することで，復号時にデータが改ざんされていないかどうかをチェックすることができます．

● データの暗号化

暗号化にはデータを暗号化するときに，前のブロックで暗号化した結果を，次のブロックの暗号化の際に利用するCBCモードを用いています．

ブロック暗号アルゴリズムを利用する場合，先頭から単位バイトずつ暗号をかけていくわけですが，平文の暗号化した結果がその後の暗号化に影響することになります．そのため，たとえば，あるバイト列が同じであったとしても，CBCモードを利用して暗号化する場合には，それ以前のデータが異なる場合には暗号文は異なる値となり，データの秘匿性を向上させることが可能となります．

▶ マルチアプリケーションとアンチブロークントランザクション

● 同時複数ファイルのオープン

接触ICカードなどでは，複数のファイルに対してアクセスする場合，

1）まず最初のファイルをOpenし，データの読み書きを行い，Closeする
2）その後，次のファイルをOpenし，データの読み書きを行い，Closeする

といった作業を行います．

非接触ICカードにおいては，カード利用者がカードをトランザクションの途中でリーダ/ライタから離してしまうことが充分考えられるため，トランザクションの最後まで処理が完了する前に通信ができなくなってしまい，データの整合性が確保できなくなってしまう可能性があります．

これを避けるために，FeliCaでは一度の相互認証で同時に複数のファイルをオープンすることで，複数のサービスファイルを1回のコマンドで同時に読み書きすることを可能としています．

また，このとき，書き込み処理の途中でどれか一つでもアクセスに失敗した場合にはすべての処理が元に戻る，という「アンチブロークントランザクション」機能を実現しています．これにより，たとえば電子マネーサービスと電子チケットサービスとを同時にアクセスする場合，電子チケット情報を書き込み，その後電子マネーの金額データを引き落とす前にカードを離されてしまうと，サービス提供者はビジネス上被害をこうむることになりますが，FeliCaの複数サービスファイル同時アクセス機能を利用すれば，このような事故を防ぐことができます．

● FeliCaOSの動作

▶ FeliCaカードリーダ/ライタ間コマンド

FeliCaでは，リーダ/ライタから送出されたコマンドをカードが受信し，カード内部で処理を実行した後，処理結果としてのデータをレスポンスとしてリーダ/ライタに対して返送します．代表的なコマンドを**表2**に示します．

リーダ/ライタは，カードが通信距離範囲内に入ってくるまでは基本的に`Polling`コマンドを定期的に送出し，周辺に通信可能なカードが存在するかどうかを確認します（**図9**）．カードは，リーダ/ライタからのコマンドをキャッチするとレスポンスを返します．これにより，カードとリーダ/ライタとの間でトランザクションが開始されます．その後，リーダ/ライタは，必要に応じて相互認証のためのコマンドやデータのRead/Writeなどを行うコマンドを送信し，カードは受信したコマンドにより処理を行います．

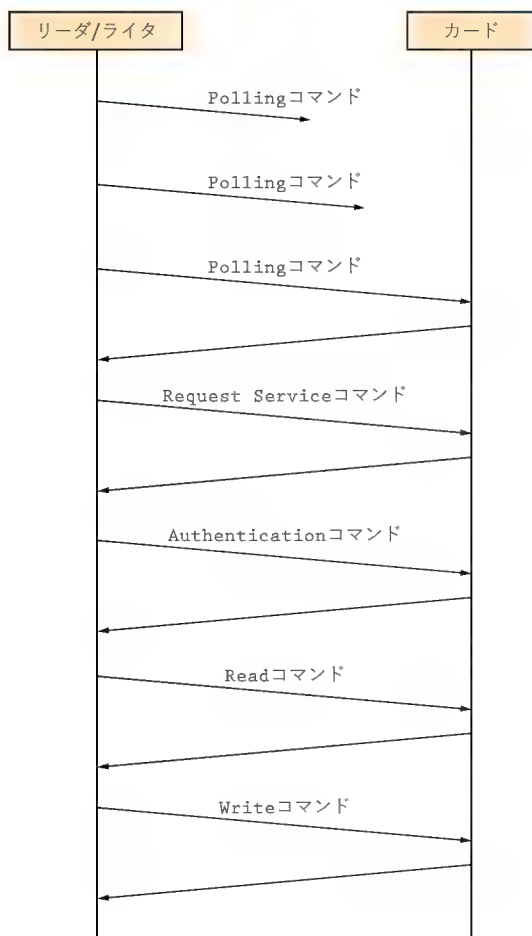
▶ カードのモード遷移

FeliCaカードは，処理の進展度合いに応じて内部的なモードを遷移させます（**図10**）．これにより，リーダ/ライタから不正なコマンドが不適切なタイミングで送られてきたとしても，処理が妨害されることを防ぎます．

電源が供給されると，カードはNeutral Modeとなります．このモードでは，カードのIDを取得するために，`Polling`コマンドを受信し実行することができます．IDを取得後，相互認証用

〔表2〕FeliCaのカードリーダ/ライタ間のコマンド

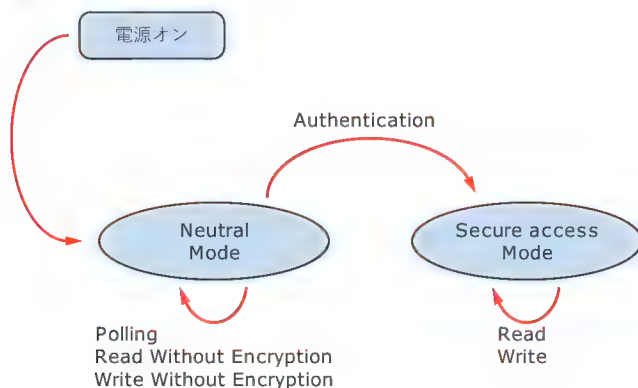
代表的なコマンド	処理内容
Polling	カードの製造IDとシステムパラメータを取得する
Request Service	サービスが存在するかどうかを調べる
Request Response	カードが存在するかどうかを確認する
Read Without Encryption	セキュリティのかかっていないサービスのデータを読み込む．相互認証は行う必要はない
Write Without Encryption	セキュリティのかかっていないサービスへデータを書き込む．相互認証は行う必要はない
Authentication	相互認証を行う．相互認証に成功すれば以降のセキュアなRead/Writeが可能となる
Read	サービスのデータをセキュアに読み込む．相互認証が完了した後に実行可能となる
Write	サービスにデータをセキュアに書き込む．相互認証が完了した後に実行可能となる

〔図9〕カードとリーダ/ライタ間の通信のようす

コマンドである Authentication コマンドを実行することで，カードは Secure access Mode へと遷移します．

Secure access Mode へ変化すると，カードは Polling コマンドを受け付けなくなります．これは，ID をすでに取得したカードが Polling コマンドに返答しないことでカードからの返答の衝突を軽減するためです．Secure access Mode では，セキュアな処理として Read コマンドや Write コマンドを受け付けることができます．FeliCa では，とくにコマンドで明示的な Neutral Mode への遷移は行わず，カードがリーダ/ライタから離されると，自動的に Mode がリセットされます．

モバイル端末向け FeliCa モジュール

ソニーでは，非接触 IC カードの「形状が自由である」という特徴を活かし，モバイル端末などの機器に対して非接触 IC カードの機能を搭載させることのできる FeliCa モジュールを現在開発しています．これは，一つのセキュアチップに外部 IC カードや外部リーダ/ライタとの通信のための無線インターフェースと端末との通信のための有線インターフェースの二つのインターフェースをもつという特徴をもちます．

〔図10〕モード遷移のようす

これにより，

- モバイル端末自体が非接触 IC カードとして利用する
- モバイル端末をリーダ/ライタとして動作させ，外部の非接触 IC カードのデータの読み書きを行う
- FeliCa モジュール内のデータをモバイル端末のからアクセスし，画面に情報表示する

といったことが可能となり，FeliCa とモバイル端末を融合させた，新しいアプリケーションが期待できます(**図 11**)．

FeliCaの開発環境

FeliCa を利用したアプリケーション開発を容易にするために，「SDK for FeliCa」という開発キットが用意されています．これは，とくにリーダ/ライタを接続したパソコンのアプリケーションを開発するためのライブラリを提供するものです．

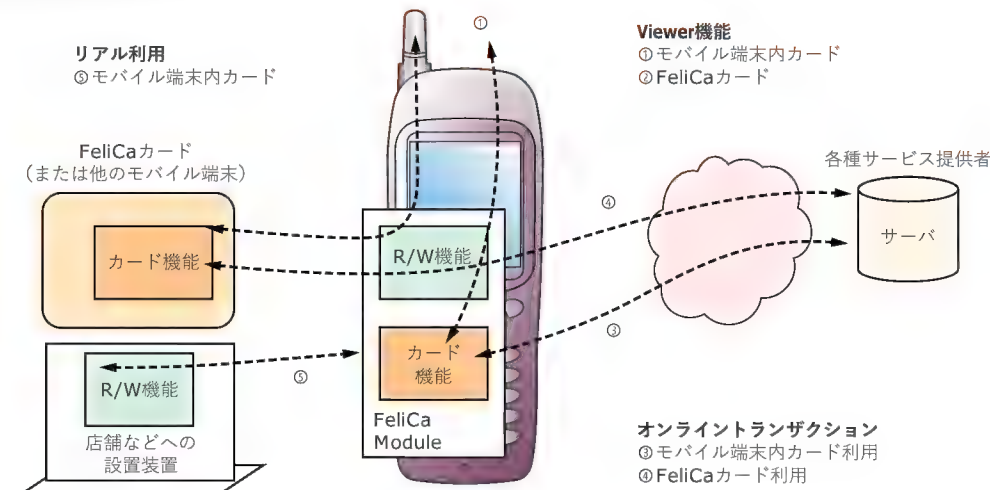
この中には，Windows 用のライブラリ(DLL)，サンプルソースコード，マニュアルなどが含まれています(**図 12**)．

交通用途での利用

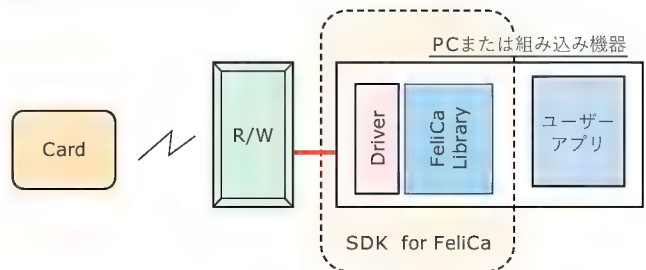
● 海外での採用事例

ソニーは，1988 年に非接触 IC カードの開発をスタートさせました．その後，1993 年に香港が中国返還と同時期に非接触 IC カード用いた交通自動料金徴収システムを導入するという計画を発表し，要求仕様が提示されました．入札の条件は①カードや改札機の耐久性を高めるため「非接触」のカードであること，②カードはバッテリレスであること，③バス，地下鉄，トラム(市電)，フェリーなどの交通用途以外にも利用できる「電子マネー」機能を保有することなどが上げられましたが，ソニーが開発中であったカードの仕様コンセプトがもっとも高い評価を受け受注に成功しました．

1995 年 11 月より実用実験を開始し，1997 年 9 月には“オクトパスカード”(**写真 2**)として稼動を開始しました．現在では 1,200 万枚以上を出荷しており，完全に香港市民の必需品として定着

〔図11〕モバイル端末向けFeliCaモジュール

しています．また，交通用途以外の領域でのサービス拡大も積極的に推進されています．たとえば，コンビニエンスストア，自動販売機，パーキングメータなどでの利用が始まっており，今後も用途はさらに広がっていく予定です．また，腕時計に非接触ICカードと同じ機能モジュールをもたせた“オクトパスウォッチ”も販売されており，形状を選ばないという非接触ならではの展開も進んでいます．

シンガポール交通局（Land Transport Authority）は，国内のバスや，地下鉄の料金徴収システムを現行の磁気カードから非接触カードへ切り替えています．ソニーは，1999年にシンガポール陸上交通局からカード約500万枚，リーダ/ライタ約2万台を受注しています．2002年の初めから本格運用が開始され，公共交通用途のほかに公衆電話やホテルの部屋鍵，IDカードなどへの展開が期待されています．

そのほかにも，中国のシンセンやインドのニューデリーなどにおいて，世界規模で“FeliCa”技術の採用が進んでいます．

〔図12〕SDK for FeliCa

● 日本国内での採用事例

日本では，JR東日本が次世代出改札システムとして，非接触式のICカードに着目し導入の検討を行っていましたが，2000年6月に国際競争入札の結果，ソニーのICカードシステムが採用

〔写真2〕オクトパスカードとオクトパスウォッチ

〔写真3〕Suica

されました．2001年4月から埼京線でモニター試験が開始され，2001年11月18日よりICカード“Suica”として首都圏の400駅以上へ導入され，本格運用が開始されました．現在では，すでに500万枚以上のカードが利用されています(**写真3**)．

“Suica”定期券は，

- 定期券にイオカード機能が付加されるので，改札機で自動的に高速で乗り越し精算が可能となり，定期券区間外の利用時でも切符を買ったり精算する必要がなくなる
- カードごとにIDで管理されるため，万が一紛失した場合でも紛失したSuica定期券を無効化し，同一内容の新規カードの再発行を可能とする

など，今までにはなかった新しいサービスが開始されています．

他の交通事業者でも非接触型ICカードの導入を予定しているところもあり，日本国内においても今後急速にさまざまな公共交通機関での非接触型ICカードの利用が進んでいくと予想されます．

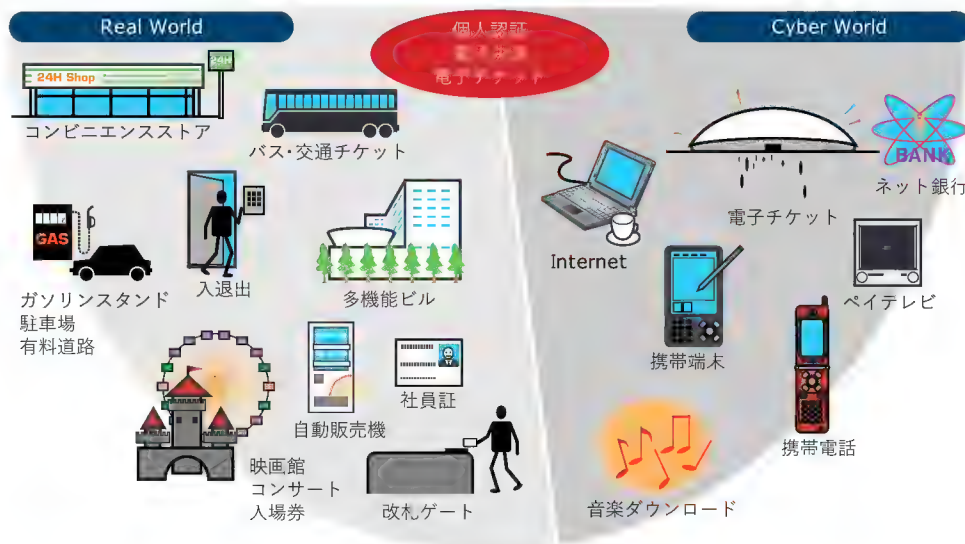
公共交通用途以外への展開

● テーマパークなどのクローズドエリアでの利用

“FeliCa”技術方式は，交通分野のほかに特定のエリア内においても数多く利用されています．

たとえば，箱根・小涌園にある温泉テーマパーク“ユネッサン”では，リストバンド型の非接触ICカードモジュールを腕に

〔図13〕非接触型ICカードの利用例

New Products ― 3Dマルチメディアコンテンツエンジンの Macintosh 版
(株)ネットディメンションは，インターネット上の3Dコンテンツを再生するWebプラグインエンジン「MatrixEngine」のMacintosh版を開発し，2003年春より提供を開始することを発表した．

付けるだけで，施設滞在中，現金でお金を支払うことなく非接触ICカードモジュールを端末にかざすだけの操作で飲食物を購入したり，施設に入場したりすることが可能で，利用者の利便性が向上しています．

また，東京・台場にあるMEGA WEBなどにおいても，飲食，物販，乗り物チケットの購入や入場確認を非接触型ICカードで行っています．ゲートシティ大崎やオーバルコート大崎といったオフィスビルでは，入退出鍵や社員証などとしても利用されています．東京三菱銀行や（株）オーエムシーカード，（株）電通などの社員証としても利用が拡がりつつあります．

このようにテーマパークや複合ビルなどさまざまな場所で，ユーザの操作性・利便性を向上させるために非接触型ICカードが利用されてきています（**図13**）．

● 電子マネーとしての利用（決済系アプリケーション）

“FeliCa”技術方式は，またビットワレット（株）が運用しているプリペイド型電子マネーサービス“Edy（エディ）”のような電子マネー分野でも広く利用されています．

非接触ICカードを利用した電子マネーは，

- 既存の磁気カードタイプのプリペイドカードと比較して偽造されにくい
- 随時入金を行うことが可能なので，1枚のカードを永続的に利用可能
- 支払いにかかる時間が大幅に短縮可能

などの特徴をもち，am/pmなどのコンビニエンスストアやレストランなどの実店舗と，インターネット上の店舗の両方で同じように利用されています．

また，FeliCaを活用した決済系のアプリケーションとしては，ソニーファイナンスインターナショナルが「eLIO（エリオ）」というネットショッピング用クレジットのしくみを提供しています．

これは，従来のインターネット上でのクレジットカード決済が，カード券面にエンボスで表記されているクレジットカード番号や有効期限の情報をパソコン上でキー入力して送信するのに対し，eLIOではeLIOを識別するID情報がFeliCaカード内にセキュアに蓄積されており，クレジットを利用する際にはリーダ/ライタ（**写真4**）にカードをかざすだけでFeliCaカード内情報がセキュアにクレジット事業者に送信されるしくみです．

〔写真4〕USB接続で使えるリーダ/ライタ

これにより，券面に表記されている個人のクレジット番号がインターネット上を流れず，第三者に悪用されにくいというメリットが生まれています．

今後の取り組み

「簡単な操作で利用可能」かつ「高いセキュリティを確保可能」な非接触ICカードは，公共交通機関はもとより，さまざまな分野でよりよいユーザーインターフェースを提供する手段として，今後ますます利用されていくと考えています．ソニーは，この非接触ICカード“FeliCa”技術方式をさまざまな機器に搭載していくことで「ユビキタス・バリュー・ネットワーク」を構築していきます．

まつお・たかし　ソニー（株）　ネットワークアプリケーション＆コンテンツサービスセクター FeliCaビジネスセンター

Information ― Symbian OSを採用したFOMA端末
富士通（株）は，シンビアン（株）が提供するSymbian OSを採用したNTTドコモ向けのFOMA端末を開発した．またシンビアンは，富士通と共同でSymbian OSの日本語対応や動画対応などのカスタマイズを行った．

ユビキタス環境のためのセキュリティアーキテクチャ

eTRONの概要

坂村　健/越塚　登

これまでにもTRONアーキテクチャに関しては，組み込み系での需要が高いITRONを取り上げてきた．今回はTRONの中でも，生活空間のあらゆる場所にコンピュータを組み込もうという考え――「ユビキタスコンピューティング」に向けたセキュリティアーキテクチャ，eTRONについて取り上げる．

eTRONは総合的なアーキテクチャであり，その実装形態としてのICカードという位置づけになっている．本章ではeTRONに関して，その概要とすでに発売されている応用製品について解説する．（編集部）

eTRONとは

●“e”とは何か

eTRONとは「オープンネットワーク対応で，セキュアな電子的『実体：entity』を多用な用途のために利用できる」ユビキタスコンピューティングのための総合的なアーキテクチャです．

eTRONの名前の由来である“e”はentity（実体）の頭文字です．このentity，じつはぴったりした日本語訳がありません．辞書には「存在」とか「実体」，「本質」というような訳が出ていますが，どうにもわかりにくいものです．

もしかしたら応用面で示したほうが，entityの意味がわかりやすいかもしれません．たとえば，電子マネーとか電子チケット，電子はんこ，電子キー，電子保険証，電子免許証……．権利・権限など，何らかの価値を含むような情報といってもよいかもしれません．たとえば，家の設備機器などが外部から勝手に制御されると困るという例では，その家の機器をコントロールできる「権限」が電子実体というわけです．

eTRONの基本的な考え方は，この電子実体を安全にあるところからあるところに受け渡すというものです．

● eTRONの提供するサービス

使い方のイメージは次のような感じです．携帯電話にeTRONが入ったとしましょう．携帯電話の中に，まさに鍵のついた入れ物がついたというイメージです．この入れ物，いわば電子金庫がeTRONだと思ってください．もちろん，この鍵は携帯電話の持ち主しか開け閉めの指示を出すことができません．

次に，この携帯電話を使ってコンサートのチケットを購入します．携帯電話をチケットセンターにつなぎ，電子的にチケットをセンターのeTRONの中から携帯電話のeTRONに安全に移動させる機能を使います．

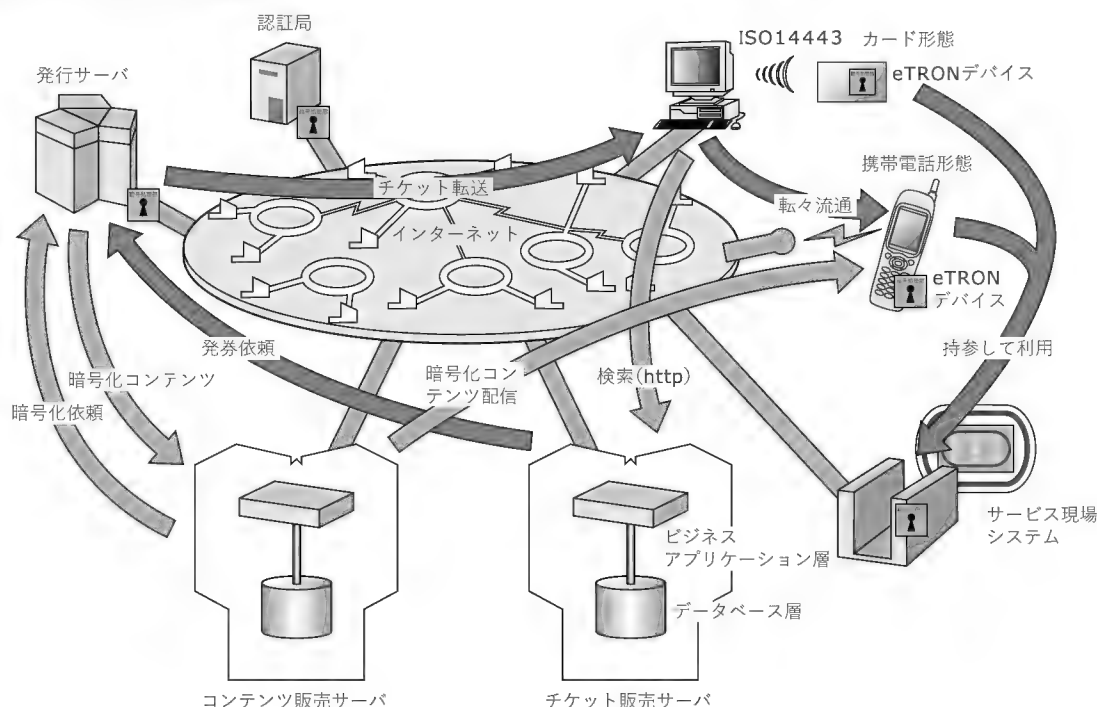
ここで，友達2人にプレゼントするつもりで3枚買ったとします．チケットセンターに接続して，チケットを購入すると，銀行からチケット代を引き落として……と，このあたりまでは今の電子商取引でもSSLなどの暗号をかければ，ある程度の安全性で行うことができます．

しかし，問題はここからチケットそのものをどう受け取るかです．ここでeTRONが出てきます．チケット，正確には「電子チケット」は安全に蓄えてあったチケットセンターのeTRON電子金庫の中からeTRONを送るための特別なプロトコル（通信手順）を使い，携帯電話中の電子金庫eTRONの中に安全に転送されます．eTRONの中から外に実体（この場合はチケット）を出すときは，データはすべて暗号化され，相手のeTRONの中に入るまで内容は（たとえ途中で盗まれたとしても）わかりません．

次に，チケットが3枚携帯電話に送られてきます．友達の一人は近くなので一緒に会場に行くことにし，eTRONの入っている携帯電話同士を向かい合わせることで，eTRONプロトコルを近接通信により行い，相手の携帯電話のeTRONの中にチケット1枚を安全に送ります．もう一人は近くにいません．そこでeTRONプロトコルをPDC（日本の携帯電話の通信方式の一つ）のデータ通信に載せて送ります．このようにeTRONはあらゆる通信手段，近接通信，PDCやPHSなどの無線通信，インターネット網とあらゆる伝送メディアが使えます．

近くの友達の都合が急に悪くなり，余ったチケットを転売することにしました．携帯電話をオークションサイトにつなぎ，eTRONモード（iモードではない！！）にして買い手を探すサイトで探し，オークション成立をリアルタイムで行い，電子商取引後，eTRONの認証通信機能により相手を認証して，同じように電子チケットを安全に相手の携帯電話のeTRONの中に送ります．さて，コンサート会場には同じくeTRONの入った電子ゲートがあります．そこで，このeTRON携帯電話をゲートにかざすことで，自動的にチケットが回収されゲートが開きます．

こういう状況で，お金やチケット（の情報）がネットワーク経由で流れていきます．このような状態を「電子実体が転々流通している」といいます．このとき重要なのが，電子実体は，相手に

〔図1〕eTRONアーキテクチャの全体像

届いたら自分のところからはなくなっている，すなわち「移動」していることです．もし，自分のところにそのまま電子実体が残ってしまったらまずいことはおわかりのとおりで，「複製」できてはいけないわけです．コンピュータはデータの複製が簡単にできてしまうわけですが，それをできないようにする，「移動」しかできないようにするというのがeTRONの本質です．

電子実体を移動するときに途中で盗聴されて複製できない，改ざんができない，また相手が正しい権利をもつ人なのかを認証するというような機能が必要になってきます．さらに，電子実体の入っている入れ物自体を分解したり解析したりウィルスを入れたりして複製や改ざんができないようになっていなければなりません．

このようなことをトータルに実現するのがeTRONアーキテクチャです．電子実体を入れる入れ物の機能をもつeTRONチップとそれをとりまくサーバが，eTRONの全体像です(**図1**)．

eTRONのセキュリティを構成する4本柱

● eTRONのセキュリティの構成

eTRONのセキュリティは，次のような四つの要素から構成されています．

1) eTRON専用ハードウェアデバイス
2) 専用デバイス間のみの転々流通機構
3) 専用デバイスAPI(Application Program Interface)
4) アクセス制御リスト

最初のeTRON専用ハードウェアデバイスは，eTRONチップ(**写真1**)を代表とする，eTRONの専用デバイスです．セキュリティはソフトウェアである程度実現できるのですが，そのソフトウェアの動作しているハードウェアを改造したり，同じ動作をするハードウェアをまるごと複製してしまうと，簡単にセキュリティが破られてしまいます．先日，ドアの錠に対してのピッキングの，新たな手法が問題になったことをご存じかと思います．その方法は錠の取りつけ部のすきまから道具を入れてかんぬき部分を直接動かしてしまうというものでした．シリンダーと鍵をピッキングに耐えられるような構造にしても，最終的なかんぬき部分を直接さわられたのではたまりません．

● さらなるハードウェア的な防御

eTRON専用のハードウェアといっても，機能的に専用のデバ

〔写真1〕eTRONチップ

イスを作っただけでは不十分です．耐タンパーでなければなりません．タンパーというのは「いじる」という意味の英語で，耐タンパーということは，改ざんや複製ができないということです．

たとえばICカードの場合，それを薬剤で解かして中のチップを見るとか，外部から特別な信号を与え不良動作をさせて中身を調べるというようなことができないような対策がされているということです．eTRONチップは耐タンパーな実装がされていて，複製や改ざん，盗聴ができないようになっています．もう少し大きなサイズのコンピュータのような形状のeTRONもあり，これは耐タンパーなケースが必要となります．

二つ目は，電子実体は上記のようなeTRON専用デバイスの間のみで渡されていくということです．これは，eTRON専用デバイスから移動する電子実体は，かならず相手のeTRON専用デバイスの中に入るということです．このため相互のeTRONデバイス同上が認証通信VPN（Virtual Private Network）を張り，つまり仮想的に二つのeTRONデバイス間に盗聴不能な専用回線がつくられ，その間を電子実体が移動します．VPNは，インターネットや無線通信のようなセキュリティの脆弱なネットワークに暗号技術を使って仮想的にセキュアな通信路を作る技術です．

三つ目は，eTRON専用デバイスの中に入っているCPUの命令セットがいわゆる汎用CPUの命令ではなくeTRON専用の命令群になっているということです．汎用のCPU命令はCPUのリソースを自由に操作できます．そうでなければ，自由に機能をプログラムで実現できないことになってしまいます．しかしeTRONのように電子実体を扱うCPUで汎用の命令が動作すると，外部から何らかの方法でプログラムを入れられると何でもできてしまいます．同じチケットを複製したり，電子マネーとして使った場合，「打ち出の小槌」になってしまいます．eTRON専用デバイスの命令は電子実体の操作が正当にできる機能を実現する必要最小限のセットとなるように定義され，これにより安全な運用ができるようになっています．

四つ目は，eTRON専用デバイスに記録される電子実体の項目ごとにどのようにアクセスするかを規定できるようになっているということです．たとえば，戸籍とか免許のような情報はそれらを発行する電子実体の発行元（たとえば役所とか公安委員会）のみが書き換え可能で，その他は読み出しのみができ，また回数券のような電子実体は減らすことのみできる，といった具合です．電子実体の性質によりアクセス制御を規定することにより，不当な操作ができないように制限を加えることができるわけです．

eTRON専用デバイスの構造

● eTRONチップとは

先にも説明したように，電子実体の発行をするようなサーバの

Column1

eTRONを応用したファイル暗号化ツール「ファイルロッカー」とeTRON/8 SDK

「ファイルロッカー」は，簡単な操作でパソコン上のファイルを暗号化して鍵をかけるツールです（**写真A**）．eTRONカードをパソコンに接続した専用のリーダ/ライタに置き，必要なファイルやフォルダをファイルロッカーアイコンにドラッグするだけで，暗号化が完了します．復号化はダブルクリックすれば直ちに戻すことができます．もちろん，権限のあるeTRONカードがリーダ/ライタに載せられていないといっさいの操作ができません．

また，ファイルロッカーは部署で利用するためアクセス権を細かく指定することができます．たとえば，課ごとにキーを設定し，同課内は暗号化/復号化できるが他の課のファイルはできないようにすることができ，さらに部長はどの課のファイルも復号化可能にするというようなアクセス管理が可能です．

このようなアクセスグループは30個まで指定できるので，いろいろなアクセス管理が可能で広い応用に対応できます．ファイルの暗号化は鍵長が256ビットのAESを利用しており，その鍵のヒントとなる情報がeTRONの電子実体として，さらにeTRON内の暗号回路で暗号化されて安全に収納されています．

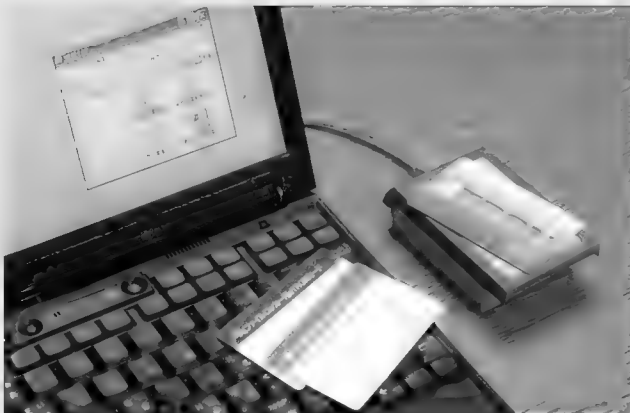
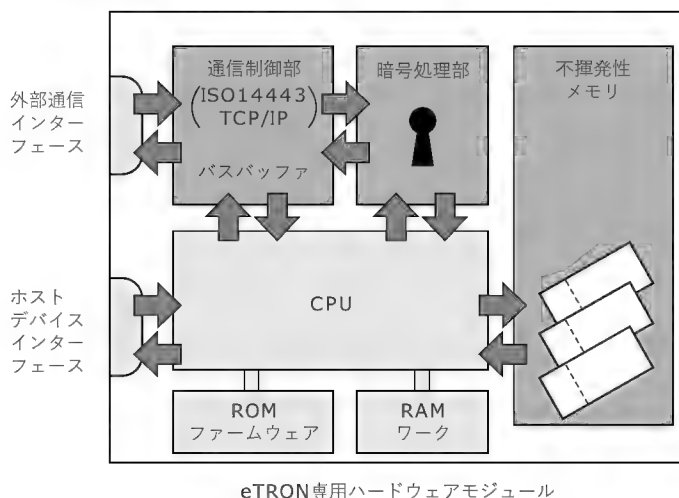
eTRONは偽造やなりすましができないので，eTRONカードの管理さえしっかりと行えば，安全なセキュリティシステムを構築することができます．パスワードを記憶しておく必要がなく，家の鍵と同様な感覚で扱うことができるので，管理が簡単です．ファイルロッカーを特定の相手と両者でもち，電子メールの暗号化に利用するという応用にも使うことができます．

パーソナルメディアでは，eTRON/8（後述）の評価や応用システムの構築に最適なeTRON/8SDK（システム開発キット）を用意しています．T-Engine（組み込み向けに標準化された共通プラットホーム）をベースに，eTRON応用システムがスクリプト言語で簡単に開発できます．

パーソナルメディア（株）
http://www.personal-media.co.jp/

〔写真A〕ファイルロッカー

〔図2〕eTRONデバイスの構造

eTRON専用ハードウェアモジュール

ようにチップの形状をとらないeTRON専用デバイスもありますが，いわゆる機器に内蔵されたり，ICカードの形状をしたeTRON専用デバイスを「eTRONチップ」と呼ぶことにします．

eTRONチップの基本構造を**図2**に示します．主要な要素はeTRON専用APIの動作するCPU，電子実体の入る不揮発性メモリ，暗号処理部，通信制御部と外部通信インターフェース，ホストデバイスインターフェースからなります．eTRONチップは無電池で動作するタイプのものや電源を外部から供給するものがありますが，電源が切れたとしても電子実体が消滅しないように保持していなければなりません．このため不揮発性メモリが必須です．暗号処理部は電子実体自体を暗号化して不揮発性メモリに格納したり，外部通信インターフェースで電子実体をやりとりする際に，認証通信やVPNを張るための機能として必要になります．

● eTRON ID

また，eTRONチップはそれぞれ固有番号eTRON IDが製造時につけられていて，個々を識別できるようになっています．eTRON IDを使って認証に利用し，相手のeTRONが誰なのかを確実に把握できるようになっています．なお，eTRONアーキテクチャではどのような暗号を使うかは規定しておらず，その識別子で暗号の種類を区別するという枠組みを規定しています．

なぜなら暗号技術は常に進化しているので，いつでも最新のものに対応できるようにすることが大事だからです．ある一つの暗号方式に拘束され，他の暗号に乗り換えができないメカニズムでは，アーキテクチャ全体が陳腐化してしまいます．ですからあくまでもeTRONとしては，枠組みを規定する形になっています．

● eTRONにおける外部との通信

eTRONと外部との電子実体のやりとりは，外部通信インターフェースという無線系の非接触型と，ホストデバイスインターフェースという接触型のインターフェースが用意されています．無線系のインターフェースは，ISO14443typeCという規格にのっとった非接触近接無線通信を基本とします．

〔表1〕eTRONの種類

種　類	概　要
eTRON/8	非接触インターフェースのみ
eTRON/16	非接触/接触インターフェース
eTRON/32	仕様検討段階
eTRON/T	非カード形状の端末

〔写真2〕eTRON/T試作機の例（YRPユビキタス・ネットワーキング研究所）

ISO14443はいわゆる非接触ICカードやRFID(Radio Frequency IDentification)の通信の規格として利用されている，キャリア周波数が13.56MHzの近接通信インターフェースです．ISO14443にはtypeA，B，Cなどの種類があり，たとえばJR東日本でタッチするだけで改札を通過でき，お金をデポジットできるSuicaと呼ばれるFeliCa(本特集第3章)規格のICカードがありますが，これはISO14443typeCの規格のカードです．

このほか，住民基本台帳で使われるICカードはISO14443 typeBを採用しています．typeAはフィリップスのMIFARE仕様として有名です．typeAは必ずしもCPUを必要としていません．typeBとtypeCの大きな違いは，typeBはマスタ/スレーブとなっていて，カード側はスレーブにしかなりませんが，typeCではマスタにもスレーブにもなるという点です．

ホストデバイスインターフェースは，たとえばeTRON機能をもつ携帯電話のようにeTRONチップを機器に内蔵する場合に，組み込み機器と通信するための接触通信ポートです．

● eTRON専用デバイスの種類

eTRON専用デバイスにはeTRON/8，eTRON/16，eTRON/32，eTRON/Tというようにシリーズ化がなされています(**表1**)．8，16，32は内蔵するCPUのビットを示していて仕様も大きくなるにしたがって高機能になります．

eTRON/TのTはTerminal(端末)のTで，カード状でない機

器の形状をしたeTRONを指します(**写真2**，前頁)．このうちeTRON/8は，すでにクレジットカードサイズのカード状のものがリリース済みで利用可能です．eTRON/8は非接触通信インターフェースのみをもち，接触インターフェースをもちません．また，それ単体でVPNを張る機能をもたず，おもにローカルで利用する目的に適します．

eTRON/16は現在試作評価中で，非接触通信インターフェースと接触インターフェースの両方をもちます．開発中のeTRON/16はクレジットカード状としても，また小さなチップの部分を切り離して使うこともできるようになっています．この部分は第三世代携帯電話に内蔵されているSIMカード(Subscriber Identify Module)という，ようするにその電話の加入権情報の入った小さなカードと同一形状をしています．

また，SIMカード状のeTRONチップは，T-Engineに備えられた専用のソケットに装着でき，これでT-EngineはeTRON/16の機能を利用できるようになります．さらに機能を強化したeTRON/32についても仕様検討を進めています．

eTRON基盤システムサーバ

eTRONアーキテクチャは，eTRONチップだけでは成立しません．実際に運用するには，**図1**に示したようなeTRONのセキュリティ基盤を構築する基盤システムサーバ群が応用に応じて必要となります．

1) 発行サーバ

発行サーバは販売サーバからの依頼で，電子実体を生成し，相手のeTRONへ確実に電子実体を配送します．このサーバはeTRONセキュリティ基盤の共用インフラストラクチャという位置付けです．

2) 認証サーバ

eTRONによるセキュリティ基盤に係わるeTRONチップ，サーバなどの証明書を管理し，第三者的に電子実体の内容を保証します．このサーバもeTRONセキュリティ基盤の共用インフラストラクチャという位置付けです．

3) 応用サーバ

応用に応じて構築される個々のビジネスサイトのサーバです．

4) 販売サーバ

コンサートチケットのようなサービス対象となる電子実体の販売をするサーバです．このサーバが直接電子実体を生成するのではなく，発行サーバに依頼し，発行と配送は発行サーバが行います．すなわち，販売サーバは電子実体の発行や配送は行いません．発行サーバへの依頼は，安全な専用回線かあるいはeTRON専用デバイスを利用してVPNで依頼します．

5) サービス現場システム

コンサート会場側の入場ゲートのようなサービス現場のシステムです．ここでは，利用者のもってきたeTRONデバイスから電子実体にアクセスします．eTRONアーキテクチャは，応用システムと階層が分かれています．すなわち，eTRON自体は決済系や販売系などの機能は含まず，価値や証明のセキュアな流通基盤を提供するのみです．また，前にも述べたように，鍵と暗号メカニズムからなる暗号基盤はeTRONの下の層として分離しています．

eTRON/8の詳細

現在利用可能なeTRON/8の詳細を説明しましょう．

eTRON/8は，2048バイトの不揮発性メモリを搭載しています．メモリは8バイト単位のブロックで管理され，全体は256ブロックからなりますが，このうち32ブロックはカードシステムが利用するため，残り224ブロックがeTRONシステムとして管理されることになります．このeTRON領域は，さらにデータ領域と定義領域に分かれています．定義領域にはeTRON IDや暗号鍵，所有者のパスワード，ファイル管理ブロック，ファイル定義ブロックが含まれます．

eTRON IDは128ビットで，製造時にユニークな番号が与えられ，書き換えができないようになっています．暗号鍵は2ブロック分用意され，内蔵のDES暗号回路を使いDESあるいはTriple-DESに対応できるようになっています．暗号鍵やパスワードは，サービス提供者がどのような設定にするかを決めるようになっています．

ファイル管理ブロックは，カードのアクセス権限，ファイル数上限を定め，ファイル定義ブロックはそれぞれのファイルの先頭アドレス，長さ，アクセス権限などが入ります．ファイル管理ブロックは，ファイル数の上限個分のテーブルとしてメモリ領域を使うので，その分電子実体が入る領域は減ることになります．

eTRON/8のAPIは，**表2**のようにシンプルです．上位システムとはセッションを張ってから通信するようになっています．セッションを構築するには2パス認証により行い，正しいカードであることを確認します．その後ファイルやレコードへのアクセスを行います．eTRONカードのリーダ/ライタとの間はISO

〔表2〕eTRON/8のAPI一覧

ニモニック	意　味
eopn_ses	Open Session
ecfm ses	Confirm Session
ecls ses	Close Session
ecre fil	Create File
edel fil	Delete File
eupd fmd	Update File Mode
elst fid	List File ID
edel rec	Delete Record
eupd rec	Update Record
erea rec	Read Record
epol car	Poll Card
eini car	Initialize Card
eupd par	Update Security Parameters

eTRON/16dual対応のリーダ/ライタシステム

Column2

大日本印刷(株)と(株)田村電機製作所はeTRON/16dualのSIMカードに対応したリーダ/ライタシステムを開発しました(**写真B**).

eTRON/16dualは(株)日立製作所製AE45XのeTRON対応品を大日本印刷がSIM化加工したもので，大日本印刷の開発したアンテナを内蔵した無線通信アダプタと呼ぶ小形のホルダに装着するとeTRONの接触端子がアンテナに接続され，非接触通信が可能となります．これにより，田村電機製の非接触リーダ/ライタでeTRONをホストと通信することができます．このリーダ/ライタはeTRONのほか，ISO14443typeAおよびtypeBにも対応しています．

また，大日本印刷(株)と十条電子(株)は，上記の無線通信アダプタにUSBインターフェースをもつ超小型ICカードリーダ/ライタを開発しました．このリーダ/ライタは非接触でISO14443通信が可能なほか，USBソケットに直接接続する接触型のリーダ/ライタとして働くことができます．eTRON/16dualにも対応しており，非接触，接触でインターフェースできるため，さまざまな場面での利用に応用できます．

大日本印刷(株) http://www.dnp.co.jp/
(株)田村電機製作所 http://www.tamra.co.jp/
十条電子(株) http://www.jujo-electronics.com/

〔**写真B**〕**eTRON/16dual対応のリーダ/ライタシステム**

14443規格の近接無線通信で行われますが，この通信も暗号処理がされています．また，リーダ/ライタ自体は暗号処理は行わず，暗号化されたまま相手のeTRONに送られます．

eTRON/16のAPI

eTRON/16では，eTRON/8の基本機能に加え，ネットワークを通じてVPNを張るための機能や，公開鍵暗号システムを利用したセキュリティシステムのインフラストラクチャであるPKI(Public Key Infrastracture)をサポートする機能をもっています．

おわりに

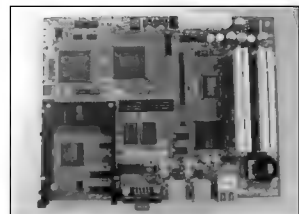
以上，ユビキタス環境構築のキーとなるeTRONアーキテクチャの概略を述べてきました．eTRONは囲みコラムにあるようにすでに基本デバイスや開発環境，応用製品がリリースされ，多様な応用の展開が始まろうとしています．

eTRONを利用してセキュリティが確保されはじめて，安心してあらゆるモノにコンピュータを入れることができるようになります．インターネットや携帯電話メールのように広がってからアタックや迷惑メールを防御するのはなかなかたいへんです．ユビキタス環境を構築する上でのセキュリティの重要さを再認識していただき，eTRONをぜひ利用していただければ幸いです．

eTRONアーキテクチャは他の多くのセキュリティシステムとは異なり，オープンポリシーをとっています．eTRONの応用製品を開発したい，あるいはeTRONアーキテクチャを構成する機器やサーバ関係で参入したいという方は，T-Engineフォーラム(http://www.t-engine.org/)にぜひコンタクトしていただくようお願いいたします．

さかむら・けん　東京大学教授/YRPユビキタス・ネットワーキング研究所所長
こしづか・のぼる　東京大学助教授/YRPユビキタス・ネットワーキング研究所副所長

Information — TIとSTマイクロ，ワイヤレスアプリケーション用オープンスタンダード設立に協力
テキサス・インスツルメンツとSTマイクロエレクトロニクスは，ワイヤレスアプリケーション用プロセッサのオープンスタンダードを共同で設定し，協力することを発表した(http://www.omapi.org/)．

汎用的な言語を用いて開発が行える

JavaCardの開発とメーラシステムへの応用

千葉新悟

ICカードのアプリケーション開発には，メモリやCPUパワーの制約などから，ハードウェアに密着した低レベルな言語が用いられることが多かった．

しかし，近年の技術の発展により，ICカードにおいても汎用的な言語であるJavaが採用される動きが出てきた．それだけでなく，VM(Virtual Machine)の概念を取り入れることにより，ハードウェアプラットホームの差異を吸収し，異なるハードウェアの上で共通のソフトウェアを実行することが可能になった．

本章では，Javaを使ったICカードアプリケーションの開発例として，ICカードのセキュリティ機能を活用したメーラシステムを取り上げる．（編集部）

近年，行政サービス，金融サービスなどで，ICカードシステムの本格導入が相次いで決定しています．2003年度の実現をめざす電子政府をはじめ，eコマースや携帯電話サービスなど，多くの分野でICカードが個人認証などのための必須ツールとして期待されています．

今回は，その中でも比較的開発が容易なJavaカードの概要と，Javaカードを使用したシステムの例としてJavaカードメーラシステムについて説明していきます．

Javaカードについて

● Javaカードとは？

これまで使われてきた接触型ICカードでは，業界や製造会社ごとに異なる仕様が存在していました．そのためカード間の互換性に乏しく，ICカードが本来もっている有効性や利便性がうまく発揮されないという問題がありました．

そこで，この問題を解消するため，ICカードに仮想マシン(VM)を搭載したものが考案され，開発されました．

● Javaカードの利点

従来のICカードアプリケーションの開発から，以下のような問題と課題がありました．

1) 各カードメーカーにより使用する関数，ライブラリが異なる
2) OSおよびアプリケーションは汎用化されたものがない
3) 各チップの固有の仕様/機能でプログラミングがされている

これらの問題を，JavaカードではICカードに仮想マシン(VM)を搭載し，Javaで開発したICカード上で動作するアプリケーション(アプレット)をカードに実装するという手法をもちいて解決しています．

● Javaカードの規格について

● ISO7816

ISO7816はISO(International Organization for Standardization)の規格検討委員会によって，外部端子付きICカードの物理的，機能的条件などについて，基本部分の互換性を確保するための国際規格として制定されました．

ISO7816は，利用分野を特定せずに適用されることを想定しており，**表1**に示すようにカードの物理的形状や，リーダライタとの通信プロトコルなどに分けて定めています．

なお，Javaカードにも従来のICカードと同様に接触型，非接触型が存在し，今回使用した接触型はISO7816に準拠しています．

非接触型はさらに密着型，近接型，近傍型に分類され，それぞれISO10536，ISO14443，ISO15963に準拠している必要があります(**表2**～**表4**)．

JavaCardAPI

Javaは，1996年にサン・マイクロシステムズにより開発・発表されたオブジェクト指向のプログラミング言語です．

この言語の特徴を使って，ICカードに適用するために，Java Card Forum[注1]が1997年4月に結成されました．その後，1997年

〔表1〕ISO7816の規定事項

ISO7816-1	Physical characteristic	材料，形状，寸法，強度および耐静電気特性を規定
ISO7816-2	Dimension and location of the contacts	外部端子の数，位置と寸法を規定
ISO7816-3	Electric signals and transmission protocols	スマートカードとリーダ/ライタ間の電気的信号の授受の手順を規定 AM1：T=1ブロック伝送方式について規定 AM2：プロトコル選択基準について規定
ISO7816-4	Use of the secure messaging	基本コマンド，ファイル構造，セキュリティの基本構造を規定

〔表2〕ISO10536の規定事項

ISO10536-1	Physical characteristic	物理的特性を規定
ISO10536-2	Dimension and location of the contacts	結合領域の寸法と位置を規定
ISO10536-3	Electric signals and reset procedures	電気信号とリセット手順を規定
ISO10536-4	Answer to reset and transmission protocols	初期応答と伝送プロトコルを規定

〔表3〕ISO14443の規定事項

ISO14443-1	Physical characteristic	物理的特性について
ISO14443-2	Radio frequency power and signal interface	電波出力と信号インターフェース
ISO14443-3	Initialization and anticollision	初期化と衝突防止について
ISO14443-4	Transmission protocols	伝送プロトコル

〔表4〕ISO15963の規定事項

ISO15963-1	Physical characteristic	物理的特性について
ISO15963-2	Air interface and initialization	非接触インターフェースと初期化について
ISO15963-3	Protocols	プロトコル
ISO15963-4	Registration of application/issuers	アプリケーション/発行者登録について

10月にはバージョン2.0がすでに仕様として制定されています．

Javaカードは複数のアプリケーションを搭載できますが，意図的に切り分けられたメモリエリア間では，不正なアクセスが行えないようになっています．そのため，それぞれを安全に動作させることが可能です．

Javaカードのアプレットの開発環境として，Javaの命令群の中からICカードで利用するのに必要なものだけをピックアップしたものが，JavaCardAPIとして提供されています．JavaCard APIのクラスライブラリはフリーで提供されているので，ユーザーはWebから自由にダウンロードして利用できます．また，作成したソースコードは，通常のJava用コンパイラでコンパイルすることが可能です．

現在は，JavaCard API 2.2が最新仕様となっています．

Javaカードのアプレットは，汎用的なJavaプログラムの開発環境を用いて開発することができます．したがって，カードメーカーだけでなく，カード発行元が自らアプレットを開発し，ユーザーのニーズに応えることができます．一度開発したアプレットは，他ベンダ製のJavaカードでも使用することができ，ソフトウェア資産を有効に活用できます．

● アプレットの作成

Javaカード内で動作するアプリケーション（アプレット）は，Java言語およびJavaCardAPIとして提供されているクラスライブラリを使用して作成します．Java言語およびJavaCardAPIの仕様書，開発キットはSunのサイトからダウンロードできます[注2]．

JavaCardAPIの現在の最新仕様は2.2となっていますが，本システムの開発時に利用したJavaカードはJavaCard 2.1 APIに準拠したカードだったため，ここではJavaCard 2.1 APIについて説明します．

● 制約

ICカードという，リソースが大幅に制限された環境で動作することから，Javaカード向けのJava言語には通常のJava言語に比べていくつかの制約があります．実際には，Javaカード上に搭載されるJVM（Java Virtual Machine）の仕様による制約ということになります．この制約について，以下に列挙します．

以下の機能はサポートされていません．

- 動的クラスロード
- セキュリティマネージャ
- ガベージコレクション
- ファイナライズ
- スレッド
- クローン
- パッケージによるアクセス制限

以下のキーワードはサポートされていません．

- `native`
- `synchronized`
- `transient`
- `volatile`

以下の型はサポートされていません．

- `char`
- `double`
- `float`
- `long`

JavaCardAPIは，通常のJava API（J2SEやJ2EE）のサブセ

注1：カード製造メーカーであるSchlumberger社とGemplus社が共同で結成したJavaCardのフォーラムのURLは，http://www.javacardforum.org/
注2：http://java.sun.com/products/javacard/

〔表5〕JavaCard 2.1 APIのパッケージ構成

パッケージ名	説　明
java.lang	Java言語のサブセットでJavaCardアプレットを開発するための基本的なクラスを提供する
javacard.framework JavaCard	アプレットのための核となる機能のクラスとインターフェースのフレームワークを提供する
javacard.security	JavaCardセキュリティフレームワークのクラスとインターフェースを提供する
javacardx.crypto	セキュリティ関連のクラスとインターフェースを含む拡張パッケージ

〔表6〕JavaCard 2.1 APIのクラス/インターフェース構成

パッケージ		クラス/インターフェース名
java.lang		
	Class	Object Throwable
	Exception	ArithmeticException ArrayIndexOutOfBoundsException ArrayStoreException ClassCastException Exception IndexOutOfBoundsException NegativeArraySizeException NullPointerException RuntimeException SecurityException
javacard.framework		
	Interface	ISO7816 PIN Shareable
	Class	AID APDU Applet JCSystem OwnerPIN Util
	Exception	APDUException CardException CardRuntimeException ISOException PINException SystemException TransactionException UserException
javacard.security		
	Interface	DESKey DSAKey DSAPrivateKey DSAPublicKey Key PrivateKey PublicKey RSAPrivateCrtKey RSAPrivateKey RSAPublicKey SecretKey
	Class	KeyBuilder MessageDigest RandomData Signature
	Exception	CryptoException
javacardx.crypto		
	Interface	KeyEncryption
	Class	Cipher

ットとなっています．Java言語による開発で一般的に使われるjava.lang.StringなどのクラスもJavaカードでは使用できません．また，java.lang.Systemクラスもサポートされていません．代わりに，javacard.framework.JCSystemクラスが使用できます．

制約およびJavaCardAPIの詳細についてはSunがリリースしている仕様書を参照してください[注3]．

● JavaCard API

アプレット開発時に使用するJavaCardAPIのうち，おもなクラスについての使用方法を説明します．

JavaCard 2.1 APIのパッケージ構成は**表5**，クラス/インターフェース構成は**表6**のようになっています．

このうち，アプレットの作成で必須となるクラスはjavacard.framework.Appletです．このクラスを継承してアプレットを実装します．javacard.framework.Appletの仕様は，**表7**のようになっています．

アプレットを作成するためには，このjavacard.framework.Appletを継承し，必要なメソッドの実装を行っていくわけですが，ここでアプレットのライフサイクルについて説明しておきます(**図1**)．

まず，アプレットがカードにインストールされます．ここでJCRE(Java Card Runtime Environment)によってアプレットのinstallメソッドが呼び出されます．そして，installメソッドではアプレットのインスタンスを生成します．このときアプレットのコンストラクタが呼び出され，ここで必要なメモリの取得などの初期化処理とregisterメソッドの実行が行われ，JCREにこのアプレットのインスタンスが登録されます．インストール処理が終了すると，アプレットは端末からの選択待ち状態となります．

端末からこのアプレットが選択されるとselectメソッドが呼び出されます．selectメソッドがfalseを返すと，JCREは端末に対してエラー(javacard.framework.ISO7816#SW_APPLET_SELECT_FAILED)を返します．アプレットの準備ができている場合はtrueを返します．その後，端末からAPDUコマンドが送信されると，アプレットのprocessメソッドが呼び出されます．processメソッドでは受信したAPDUに応じた処理を行い，応答を返します．

端末が別のアプレットを選択すると，deselectメソッドが呼ばれ，このアプレットは選択待ち状態に戻ります．

注3：JavaCard言語仕様はJCVMSpec.pdfに，JavaCardAPI仕様はJavaCard21API.pdfに記述されている．

〔表7〕javacard.framework.Appletクラス

メソッド名	説明
protected Applet()	アプレットオブジェクトを生成するためinstallメソッドから呼び出される
void deselect()	別のアプレットが選択されたときにJCREから呼び出さる
Shareable getShareableInterfaceObject(AID clientAID, byte parameter)	このサーバアプレットから共有可能なインターフェースオブジェクトを得るためにJCREから呼び出される
static void install(byte[] bArray, short bOffset, byte bLength)	アプレットのインスタンスを生成するためにJCREから呼び出される
abstract void process(APDU apdu)	APDUコマンドの受信時にJCREから呼び出される
protected void register()	アプレットのインスタンスとAIDをJCREに登録するために呼び出される
protected void register(byte[] bArray, short bOffset, byte bLength)	アプレットのインスタンスと指定されたAIDをJCREに登録するために呼び出される
boolean select()	このアプレットが選択されたときにJCREから呼び出される
protected boolean selectingApplet()	このアプレットが選択されているかどうかを返す

アプレットの実装

前述のとおり，アプレットを作成するにはjavacard.framework.Appletを継承します．このとき，実装する必要がある最低限のメソッドは，

- コンストラクタ
- installメソッド
- processメソッド

の三つだけです．

1）コンストラクタ

コンストラクタでは，必要なメモリの取得や初期化といった通常のクラスの実装で行う処理とregisterメソッドの呼び出しを行います．ここで取得したメモリの内容は，Javaカードの電源が切られた場合でも保存されます．

2）installメソッド

アプレットがJavaカードにインストールされたときに1度だけ実行されるメソッドです．このメソッドでは，このアプレットのインスタンスを生成します．

3）processメソッド

端末から送信されたAPDUコマンドを受信するメソッドです．ここでコマンドに応じた処理を行い応答を返します．

受信したAPDUコマンドは，パラメータapduで渡されます．apduは，APDUを表現したjavacard.framework.APDUクラスのインスタンスです．受信したAPDUコマンドのバイト数を取得するには，APDU#setIncomingAndReceiveメソッドを使用します．APDU#getBufferメソッドにより受信したAPDUコマンドのバイト列を取得します．

APDUコマンドのバイト列からAPDUの各フィールドを参照する場合，各フィールドのオフセット値がjavacard.framework.ISO7816クラスに定義されています．ただし，定義されているのは，フィールド位置が固定となるCLA～データの先頭位置まででであり，Leについてはデータ部のサイズにより位置が可変となるため定義されていません．Leの値は，APDU#setOutgoingメソッドで取得できます．

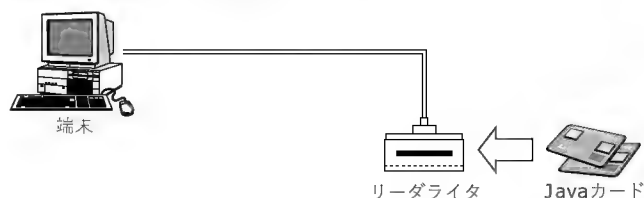
〔図1〕JavaCardアプレットのライフサイクル

応答を返すには，APDU#setOutgoingLengthメソッドで応答のバイト数を設定し，APDU#sendBytesメソッドでAPDUクラスのバッファを送信するか，APDU#sendBytesLongメソッドにより指定したバッファを送信します．

処理中にエラーが発生した場合は，javacard.framework.ISOExceptionをスローします．例外のスローは，throwではなくISOException#throwItメソッドを使用します．例外の要因としてjavacard.framework.ISO7816で定義されている値をISOException#throwItメソッドのパラメータとして渡します．

● Javaカードへのインストール

アプレットは，通常のJavaコンパイラでコンパイルできます．

〔図2〕一般的なシステム構成

その際，JavaCardAPIのクラスライブラリのパスをクラスパスに指定します．

今回使用したJavaカードへのインストールには，Javaカードのベンダから提供されたツールを使いました．このツールではJavaカードへのインストールの前にアプレットのクラスファイルから別のファイル形式への変換が必要でした．

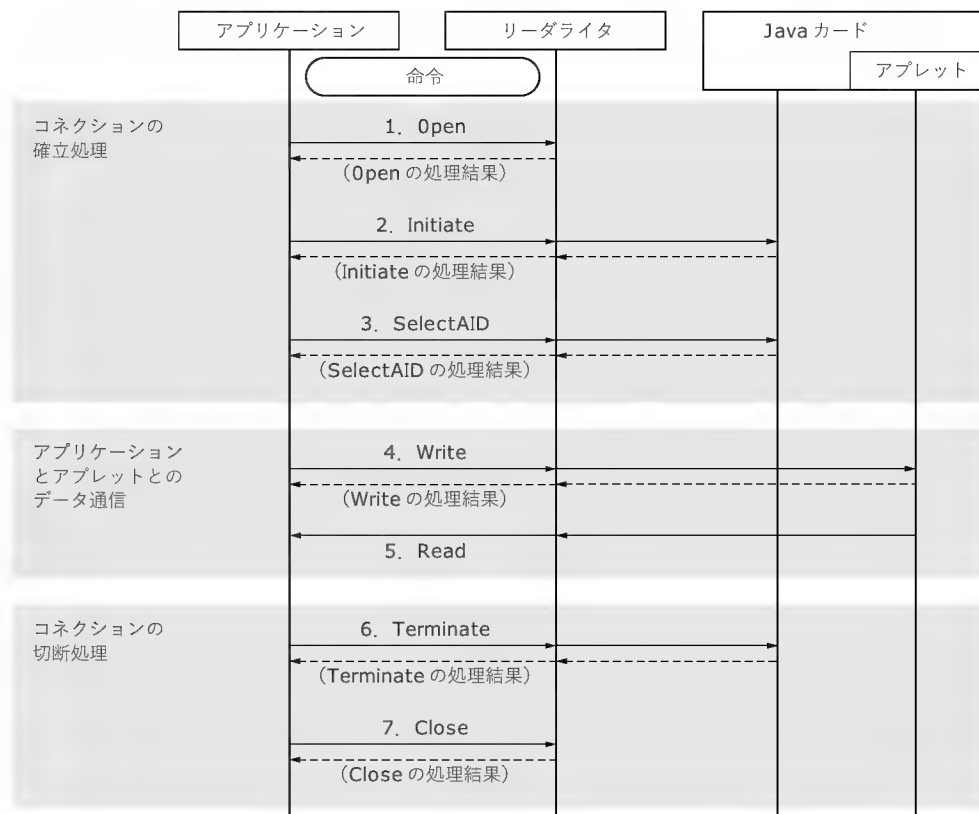
JavaCard 2.2開発キットには，ファイル形式の変換ツールやスクリプト生成ツール，APDU送信ツールなどが付属しています．そのためベンダ固有のツールや開発環境は必要なく，Javaカード，リーダライタ，リーダライタのドライバさえあればアプレットの開発が可能となっているようです．これについては，今後検証したいと思います．

● アプリケーションとアプレットとの通信手順

通常，端末上で動作するアプリケーションがJavaカード上で動作するアプレットに対して命令を送信，アプレットからの処理結果を受信するためには，リーダライタとの間でコネクションを確立する必要があります．また，アプレットとの送受信が完了した後，先に確立したコネクションを切断する必要があります．**図2**にアプリケーションおよびアプレットを含む一般的なシステム構成図を，**図3**に確立方法および切断方法のシーケンスを示します．

図3に記述している命令の内容は，下記のとおりです．

1）`Open` ——リーダライタとのコネクションを確立します
2）`Initiate` —— Javaカードの電源をONにします（Javaカードの VMが起動する）
3）`SelectAID` ——アプレットをカードにロードしたときに指定した`AID`(Applet IDentifier)を送信します．この処理によって，Javaカードに複数搭載されているアプレットのうち，特定のアプレットに対して命令を送信することが可能になります
4）`Write` ——アプレットに対して命令を送信します
5）`Read` ——アプレットからの処理結果とは別に，`Write`で要求したデータを受信します
6）`Terminate` —— Javaカードの電源をOFFにします〔Javaカードの VMが停止する．別のJavaカードに差し替えて通信を継続するような場合は，この命令の送信後，2)の処理を行うことによって通信を継続することが可能〕
7）`Close` ——リーダライタとのコネクションを切断します〔再

〔図3〕コネクションの確立および切断のシーケンス

度，アプレットと通信を行う場合は，1)の処理を行う必要がある〕

Javaカードのメーラについて

われわれが考えるJavaカードシステムとは，Javaカード内にすべてのデスクトップ環境を保持することにありますが，今回はJavaカードを使用したシステムの一例として，メーラシステムを紹介します．

● Javaカードメーラシステムの概要

Javaカードメーラシステムは，S/MIMEを使用することによってメール自体のセキュリティ対策を行っています．S/MIMEの概要については後述します．

● システムの運用

Javaカードメーラシステムは，カード発行システムとメーラシステムを組み合わせて構築されています．

まず，カード発行システムを使用して利用者が使用するカードに必要な情報を登録して利用者にカードを発行します．カード利用者は発行されたカードを使用することにより，サービスを利用することができます(**図4**)．

● システム構成

1) カード発行システム

カード発行システムは，**図5**に示すように認証局への証明書の発行を申請し，証明書を取得します．

その後，利用者の証明書，メールのアカウント情報およびJavaカードメーラアプリケーションをJavaカードに登録します．

2) Javaカードメーラ

Javaカードメーラは，カード発行システムが発行したカードに登録されている利用者の情報を使用してメールの送受信を行う専用のアプリケーションです．

また，メールの暗号化を行い，安全なメールの送受信を実現しています．

メール送受信の手順は，以下のようになっています．

2.1) メールの送信

メール送信時は，以下の手順で送信処理を行います(**図6**)．

① メール情報取得

Javaカードから利用者のメールアドレス，パスワード，メールサーバのアドレスを読み込みます．

② 送信先公開鍵を取得

認証局から送信先の公開鍵証明書を取得します．

③ 電子署名の生成

④ メールの暗号化

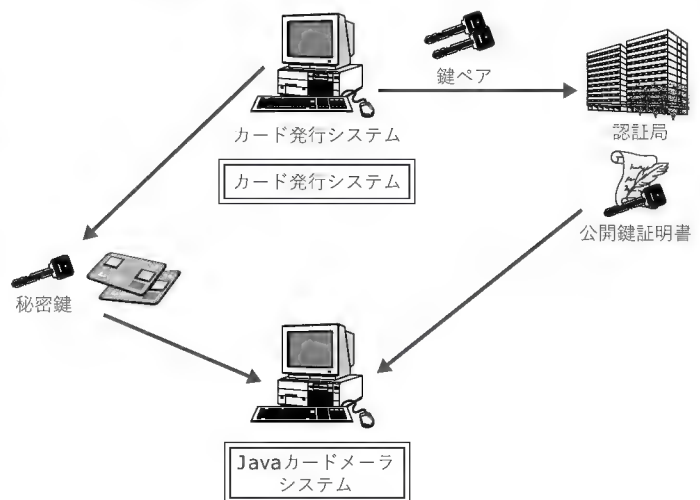
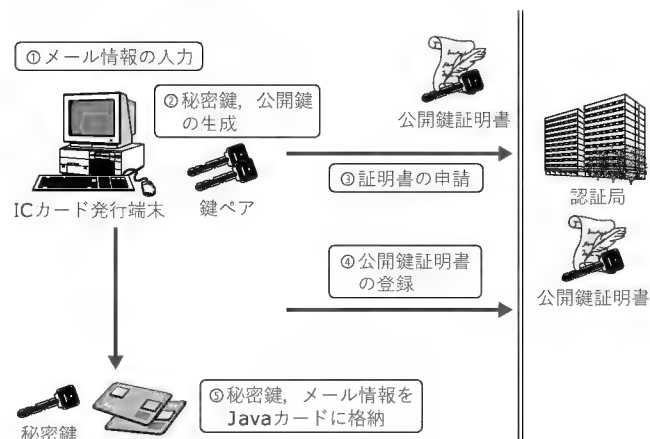
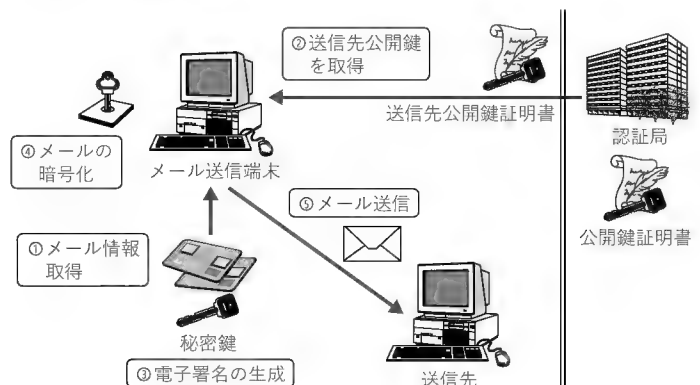
図7の手順にしたがって，送信データを暗号化します．

⑤ メール送信

①～④で作成したデータを送信します．

2.2) メールの受信

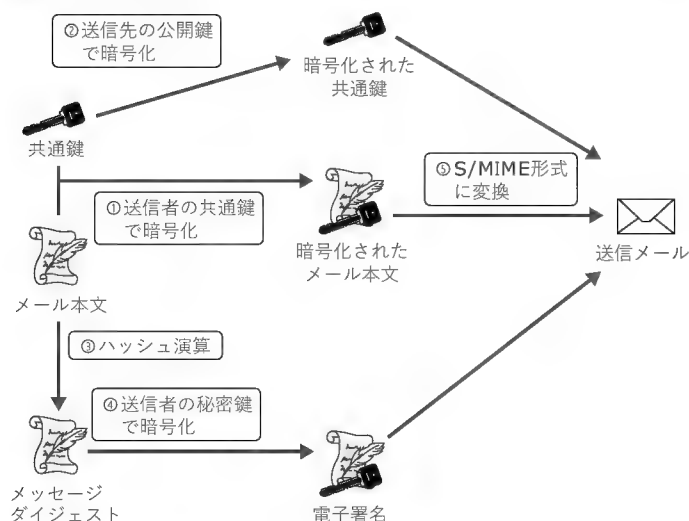
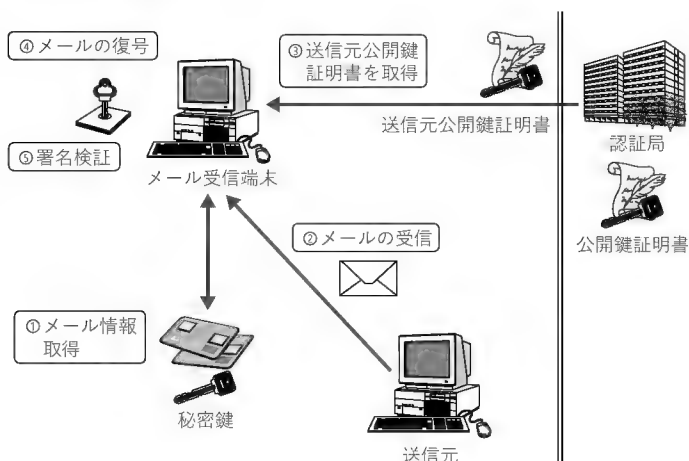
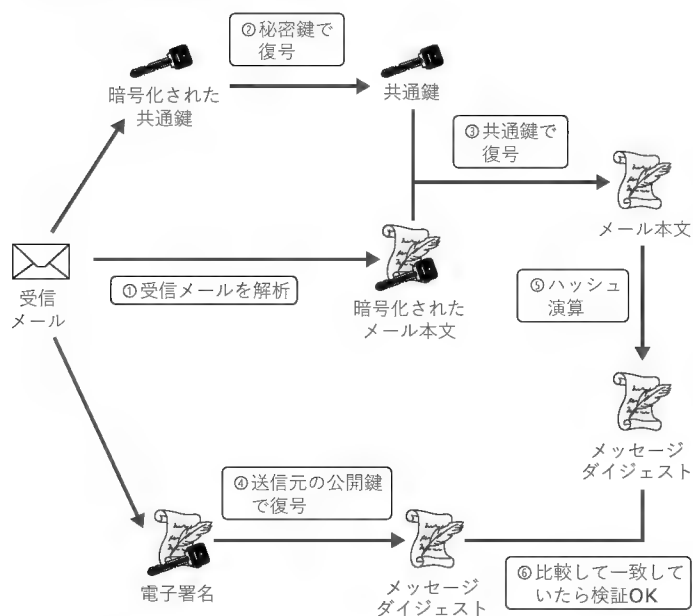
メール受信時は，以下の手順で受信処理を行います(**図8**)．

〔図4〕システム構成

〔図5〕ICカード発行システム

〔図6〕メール送信

① メール情報取得

Javaカードから利用者のメールアドレス，パスワード，メールサーバのアドレスを読み込みます．

②メール受信

メールを受信します．

〔図7〕暗号化の手順

〔図8〕メール受信

〔図9〕復号化の手順

③送信元公開鍵証明書を取得

認証局から送信元の公開鍵証明書を取得します.

④メールの復号

⑤署名検証を行います

図9の手順にしたがって，受信データの復号および署名検証を行います.

S/MIMEについて

● S/MIMEとは？

S/MIMEは，暗号化と電子署名の機能を用いて**表8**に示す基本的なセキュリティサービスを提供することによりネットワークの脅威を防ぎ，安全なセキュリティインフラを構築します[注4].

● S/MIMEの関連技術

S/MIMEはセキュリティインフラの構築を，以下の技術を組み合わせて使用することで実現します.

1）公開鍵暗号方式

公開鍵暗号方式は，対になっている二つの暗号鍵を用います.この二つの鍵はそれぞれ「公開鍵」と「秘密鍵」と呼び，対となる二つの鍵の組み合わせを「鍵ペア」と呼びます．公開鍵暗号方式は，「片方の鍵（公開鍵）で暗号化した情報は，もう片方の鍵（秘密鍵）でないと復号できない」という性質をもっています.

この性質を用いることで，安全にメールを送信することができます.

以下に具体的な処理手順を記述します.

メッセージを送信する人を「Aさん」，受信する人を「Bさん」とします.

① Bさんは秘密鍵と公開鍵のペア（鍵ペア）を生成する

② Bさんは，秘密鍵は誰にも知られないよう厳重に保管し，生成した公開鍵はネットワーク上で公開する

③ Aさんは，公開されているBさんの公開鍵を取得する

④ Aさんは，取得した公開鍵を使用してメッセージを暗号化し，Bさんに送信する

⑤ Bさんは，自分の秘密鍵を使用して暗号文を復号する（**図10**）

公開鍵で暗号化した暗号文は，対応する秘密鍵を使用しないと復号できないため，第三者が公開されているBさんの公開鍵を使用しても暗号文は解読できません.

秘密鍵が外部に漏れないかぎり，暗号文は秘密鍵をもつBさんのみが復号できます.

しかし，公開鍵暗号方式は複雑な演算処理を行うため処理に時間がかかってしまいます.

そこで高速な処理を可能にするために，公開鍵暗号方式と次に説明する共通鍵暗号方式を組み合わせて使用します.

1）共通鍵暗号方式

暗号と復号に同じ鍵を用いる暗号方式を，共通鍵暗号方式と

注4：http//www.ipa.go.jp/security/pki/072.html

〔表8〕S/MIMEにおける基本的なセキュリティサービス

種　別	説　明
守秘性(Confidentiality)	あるデータを意図した特定の相手のみが読めるようにすることができる
認証(Authentication)	送信元を証明することができる． 認証には目の前の相手を確認するローカル認証と離れた相手を確認するネットワーク認証がある． PKIはネットワーク認証において高レベルな認証を実現できる． 認証は電子署名の機能を用いて実現し，不正侵入やなりすましの脅威を防ぐ
完全性(Integrity)	あるデータが変更されていないことを証明する． 電子署名の機能を用いて実現し，改ざんの脅威を防ぐ
否認防止(Non-Repudiation)	ある人が以前に行った行動を証明し，否認の脅威を防ぐ． 認証と完全性が正しく機能することで実現される

呼びます．また，共通鍵暗号方式で使用する鍵を，共通鍵と呼びます．

以下に具体的な処理手順を記述します．

メッセージを送信する人を「Aさん」，受信する人を「Bさん」とします．

まず，Aさんは以下の手順でメッセージを送信します．

① 共通鍵を生成する
② 安全な方法を用いて，生成した共通鍵をBさんに送信する
③ 生成した共通鍵を使って，Bさんに送信するメッセージを暗号化する
④ 暗号化したメッセージをBさんに送信する

次に，Bさんは，以下の手順でメールを受信します．

⑤ Aさんから暗号文を受信する
⑥ 事前に，安全な方法で受領した共通鍵を使用して暗号文を平文に復号する(**図11**)

ここで問題となるのが，AさんからBさんに安全な方法で共通鍵を送信する手段です．

メッセージを暗号化しても，第3者に暗号化したメッセージと共通鍵が盗まれてしまうとメッセージが盗聴されてしまいます．そこで前述の公開鍵暗号方式と組み合わせて，メッセージの暗号化は処理が高速な共通鍵暗号方式で行い，メッセージよりもデータが小さい共通鍵を公開鍵暗号方式で行うことにより，高速で安全にメールの送受信を行います．

3）電子署名

電子署名は，メッセージに対してメッセージの作成者を特定するための署名を付与します．

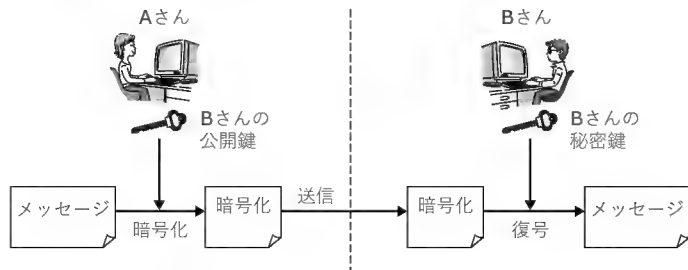
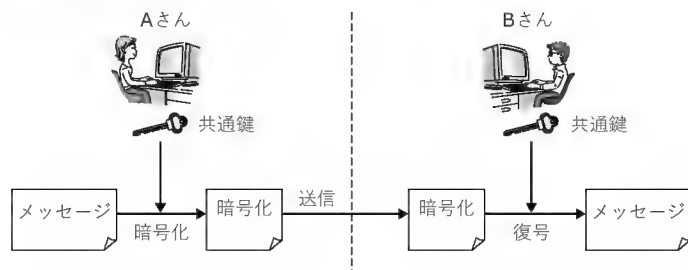
電子署名は，メッセージに対してセキュアハッシュ関数を使用してハッシュ演算を行うことでダイジェストを生成し，さらに生成されたダイジェストを秘密鍵で暗号化することで生成されます．

電子署名を用いると，メッセージの完全性と作成者の認証が可能になります．電子署名を生成することを署名と呼び，電子署名が有効であることを確認することを署名の検証といいます．

以下に，具体的な処理手順を記述します．

ここでメッセージを送信する人を「Aさん」，受信する人を「Bさん」とします．

●電子署名の生成(Aさん)

〔図10〕公開鍵暗号方式

〔図11〕共通鍵暗号方式

① 署名したいメッセージから，ハッシュ関数を使ってダイジェストを生成する
② 生成したダイジェストを自分の秘密鍵で暗号化する
③ メッセージと生成した署名をBさんに送信する

●電子署名の検証(Bさん)

④ 受信したメッセージから，ハッシュ関数を使ってダイジェストを生成する
⑤ 受信した電子署名を，Aさんの公開鍵を使って復号する
⑥ ④において生成したダイジェストと，⑤で復号したダイジェストを比較し，完全に一致することを確認する(**図12**)

電子署名は，メッセージのダイジェストを署名者の秘密鍵で暗号化したものです．検証者は，まず署名者の公開鍵を使って電子署名をダイジェストに復号します．これでダイジェストを生成した人がたしかに署名者であることを確認できます．

次に，送られてきたメッセージのダイジェストを生成し，電子署名から復号したダイジェストと比較することによって，メッセージが改ざんされていないことを確認できます．

電子署名の検証が成功すると，以下のことが確認できます．

① メッセージが改ざんされていないこと

② メッセージは，検証に使用した公開鍵と対になる秘密鍵によって署名されたこと

ダイジェストが一致せずに検証が失敗すると，以下のいずれかの事象が発生したことを確認できます．

① メッセージが改ざんされたこと
② 電子署名が改ざんされたこと

おわりに

今回は，Javaカードの実用例として，Javaカード自体のメモリ容量が少ないこともありメーラシステムの構築にとどまりましたが，今後はJavaカードにデスクトップ環境すべてを保持するべく，その他のアプリケーションの開発も行っていきたいと考えています．

また，Javaカードを使用したシステムとしては，第三者にカードが使用されないように個人認証も必要不可欠だと考えていますが，今回は実装できませんでした．この点も今後の課題として残っています．

ちば・しんご　(株)ネビット

〔図12〕電子署名

Aさん
Bさん
メッセージ
メッセージ
署名
送信
メッセージ
署名
メッセージ
ハッシュ演算
ダイジェスト
Aさんの秘密鍵
暗号化
署名
ハッシュ演算
ダイジェスト
比較
ダイジェスト
復号
署名
Aさんの公開鍵

Information ── 無料のマイクロコントローラ・デザインキットPicoBlazeのサポート範囲が拡大
ザイリンクスは，同社が提供する無料の8ビットマイクロコントローラ・デザインキットPicoBlazeのサポート範囲をVirtex-II FPGAとCoolRunner-II CPLDまで拡大すると発表した．

ICカードの未来へ向けて

次世代スマートカードの技術と応用

大山永昭

ICカードは実用技術として，すでに普及している．しかし，ICカードのもつポテンシャルは高く，将来的にさらなる応用が期待される．そして，さらなるセキュリティの強化も必要である．

次世代スマートカードシステムの開発と普及を目標として設立された次世代ICカードシステム研究会(NICSS)は，ICカード社会の将来に向けての研究を進めている．

そこで，本特集の最後にICカードとICカード社会の未来像について，現状との比較をまじえつつ解説する．

(編集部)

ICカードは，1970年代に試作されて以来，欧州においては銀行やクレジットなどの決済用カードやプリペイド型のテレホンカードとして実用化されている．我が国においても，保健医療カードや商店街などの各種のポイントカード，さらには近年開始されたICテレホンカードなどへ応用が進んでいる．これらの利用方法のほとんどは，電子マネーに代表されるように電子的なバリュー(価値のあるもので，電子マネーの残金や保健医療カードの保健医療情報などの実体となる情報)そのものを記録し，カード利用者が携帯するオフライン方式である．

一方，インターネットの爆発的な普及により，1990年中頃から電子政府や電子商取引などの先進的な情報システムの構築が世界規模で開始されている．これらの情報システムは技術的なインフラのみならず，従来の自筆の署名や記名捺印を電子化する手段を必要とするため，そのもっとも有効な実現手段として電子署名が注目されている．その結果，現在までに我が国を含む欧米アジアなどのIT先進国では，電子署名の法的な有効性を認めるための新法を制定し，技術的なインフラを整備するためにPKI(Public Key Infrastructureの略．公開鍵暗号方式を用いた電子署名手法のインフラ化を意味する)を推進している．

本章では，社会のIT化にともなって変化する環境において，PKIとともに電子空間の安全性確保に不可欠なICカードについて解説する．はじめに，従来型ICカードの基本動作とファイル構造などについて説明し，次にIT社会におけるスマートカード(後述)の役割を明らかにする．そして，この役割を担うために開発された次世代スマートカードと，その運用・管理手法および最大の特徴である2階層のPKIを解説する．さらに，次世代スマートカードが開発されてきた歴史と住民基本台帳カードなどへの応用について触れる．

ICカードの種類と特徴

ICカードは，キャッシュカードやクレジットカードの大きさのカード内に，ICチップを埋め込んだものである．その種類は，中央演算装置(CPU)およびインターフェースの接点の有無により4種類に大別される．CPUの有無は，ICカードの機能に大きく影響するため，CPU付きを単なるメモリカードと区別するために，欧米などでは一般的に「スマートカード」と呼んでいる．また，接点のないICカードには，その動作範囲がリーダ/ライタとの距離により密着型(～約3mmまで)，近接型(～約10cmまで)，近傍型(～1mまで)，遠隔型(数m)の4種類がある．

ICカードがもつ機能をその構造で分類すると，

① メモリカード

② ワイヤードロジック付きICカード

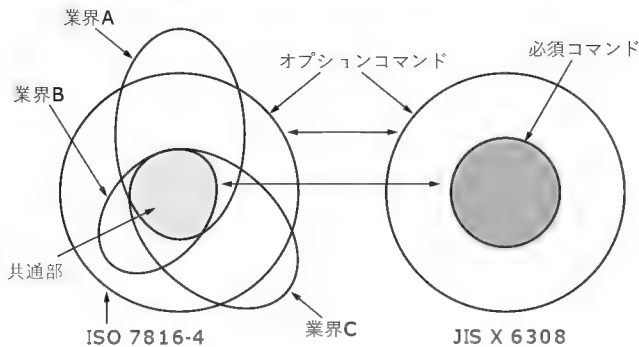
③ スマートカード

に分けることができる．

①のカードは，近年ではノート型パソコンなどに用いられるPCMCIAのメモリカードと同じ機能であるが，主として財布に入れることや券面表記の利便性から，クレジットカードなどと同じカード形状をしている．このようなカードは，単なるメモリになっているため，記録されたデータの安全性などの確保や正当な利用者以外の読み出しを禁止することなどのアクセス制御を行うには，記録されるデータの暗号化などの対策を講じて用いる必要がある．

②のカードは，①のカードに比較的簡単なロジックをハード的に付加したものである．この種のカードでもっとも有効なのは，DESなどの暗号ロジックをカード内に記録することで暗号演算をカード内で高速に行うことである．こうすることで，秘密鍵共有暗号方式を用いた相互認証(一般的にはカードと端末)によるアクセス制限と記録されるデータの暗号化が可能になる．

③のカードは，もっとも柔軟性に富んだものであり，その一般的な構造は，CPU，ROM(カードOSなどの基本ソフトウェアを記録するメモリ)，RAM(カード内での演算処理の途中結果などを記録するメモリ)，不揮発性メモリ(ユーザーがカードに記録するデータやソフトウェア用のメモリで，電源が切れても

〔図1〕ISO規格とJIS規格の関係を示す概念図

消えることのない書き換え可能なもの．EEPROM，フラッシュメモリ，FRAMなどがある)，コプロセッサ(主としてRSAなどの非対称鍵暗号処理を高速に行うための補助演算装置で，一般的にはオプションである)からなり，まさに小さな専用パソコンである．さらに，このCPU付きのスマートカードは，近年，著しい技術進歩を遂げており，たとえば，MULTOSやJavaCardと呼ばれるものは，それぞれのカードがサポートする高級言語で書かれたソフトウェアを，カードを発行した後でも追加することができる仕様になっている．

これらのカードはすべて外部から電源を供給して動作するため，電池は積んでいない．また，カードに埋め込まれているICチップはすべて数ミリ角以下の大きさであるが，その価格はまだ比較的高価であるとともに，機能に比例してさらに高くなる．

その主たる理由は，これまでに実用化/導入されたICカードのほとんどが多くて数十万枚の単位であることから，大量生産によるコストダウンができていないためである．そのため，各アプリケーションで用いるカードは，そのアプリケーションに特化させることにより，すなわち不要な機能を削ることでカードの単価を下げるのが一般的であった．結果として少量多品種生産となり，大量生産に結びつかなかったわけである．

国際標準規格

これまでにISO-IEC/JTC1/SC17により制定されている国際標準規格には，カードの形状，物理特性，リーダ/ライタとの通信プロトコル，カードへのコマンドなどがあり，カードの普及に大きく貢献してきている．ここでは，ISO7816-4などで規定されるカードへのカードコマンドを中心に説明する．

カードコマンドは，カードの発行やアプリケーションの追加などに用いるシステムコマンドと，アプリケーションが乗せられた後に使うユーザーコマンドに分けられる．歴史的には，磁気カードなどの各種カードの利用は，発行者の専用システムで行われてきたため，ユーザーコマンドから国際標準化が行われた．その後，発行コマンドについても，後述する認証などの手順を別にして標準化が行われた．

いうまでもなく既存のカードのほとんどは，カード発行者が所有し，その利用をカード利用者に許可するという運用形態をとっているため，カードに記録されるファイルの追加や削除などの処理は，カード発行者が管理/運用する専用システムにより行われている．そのため，カード発行時の認証などの手順は，依然として国際標準になっていない．また，ユーザーコマンドについても，国際標準規格ではすべてのコマンドをオプションとして定義している．

逆の言い方をすると，国際規格で規定されているコマンドを一つでもサポートすれば，そのカードはISO準拠ということができる．その結果，ISO準拠のカードであっても，一般的にはカードの相互運用性を確保するには，そのカードがどのコマンドをサポートしているかについて，十分な注意が必要である．

そのため，JIS X 6308では，国際標準で規定されているコマンドセットから，スマートカードとしての利用価値の高いいくつかのコマンドを必須として定義している．ISO規格とJIS規格の概念的な関係を**図1**に示す．したがって，JISに準拠するすべてのカードは少なくとも必須コマンドをサポートしているので，異なるメーカーのスマートカード間の互換性を維持することができる．

近年は，システムの安全性をより向上させるために，パスワードの代わりに指紋などのバイオメトリクスを用いる手法が国際標準化の対象として提案されている．この手法では，カード内にバイオメトリクス情報を記録するだけで照合演算をカード外部で行うものと，照合演算そのものもカード内で行うものとがある．さらに，指紋をカードに組み込んだ読み取り装置により行うものも試作されている．これらに関する標準化作業はまだ準備段階だが，今後，きわめて重要な課題になると予想される．また，後述する次世代スマートカードが用いる2階層のPKIについても，現在，国際標準にするための準備が行われている．

従来型スマートカードの論理構造

従来のスマートカードは，カードと送受信するためのコマンドおよびその引き数を解釈し実行する一種のインタプリタ形式のOSと，**図2**の概念図が示すファイル構造から成り立っている．**図2**に示すファイル構造はあくまで一例であるが，この例ではカードに対して最初にアクセスするファイルをCDF(Card Domain File)としている．一般的には，このような構造をもつカードでは，このファイルにカード発行者，カード管理番号，利用者の氏名などの基本情報を記録する．そしてCDFの下には，複数個のDF(Dedicated File)とEF(Elementary File)を設置することができる．また各ファイルには，アクセス制限のための鍵を設定することができる．

DFはアプリケーション単位で設定されるのが一般的であり，その指定はAID(Application ID)と呼ばれる番号を用いて行われる．このAIDは，唯一性を確保することを目的として国際標準により規定されており，国際的に利用されるクレジットカードなどでは，国際利用のAID登録機関としてオランダのKTASが，

〔図2〕従来型スマートカードのファイル構造を示す概念図

〔図3〕保健医療カードに記録される情報の一例を示す概念図

〔図4〕設計された保健医療カードの構造

また国内利用には各国に登録機関(日本では，INSTAC)が設置されている．

またEFは，CDFあるいはDFの配下に設定されるもので，その指定は任意の番号(0から15までが基本)で指定する．このEFでたいせつなのは，アクセス制御で重要な鍵も一つのEFとして記録されることである．さらに各ファイルには，EF番号で規定される鍵の照合結果の組み合わせにしたがってアクセス制御がなされる．以上のことをよりわかりやすく説明するために，以下では医療データの記録を例として取り上げる．

いま，**図3**に示す簡単なカードを例として，カードの構造とアクセス制御のメカニズムを解説する．**図3**に示すカードには，患者の基本情報，救急情報，健康診断結果などが記録されているとする．カードシステムを設計するときにもっとも重要なことは，アクセス制限を含むデータ保護のポリシーの策定だが，ここでは簡単にするために以下のとおりとして話を進める．

1) 患者の基本情報は，カード発行者により記録するが読み出しは自由とする．ただし，カード利用者本人の確認を行うためにPIN(Personal Identification Numberの略．一種のパスワード)の照合を行う
2) 救急情報は，本人が意識不明になることもあるので，PINの照合は用いないが，救急隊員などの医療従事者のみが読み出せるものとする
3) 健康診断結果は，PIN照合と医師の資格を認証することにより読み出せるものとする

なお，ここでは救急情報と検診結果の書き込みに対するアクセス制限は省略する．

上記のポリシーにしたがってファイルを設定することを考える．基本情報についてはCDFに，患者のPINをEF0に，またCDFへの書き込み制限のための鍵をEF1に記録する．次に，救急情報ファイルとしてDF0を設定し，その配下に救急情報を実際に記録するEF0と，その読み出しを制限するための鍵を記録するEF1を設定する．そして，健康診断結果を記録するDF1とその配下にデータを記録するEF0，PINを記録するEF1，医師の資格認証用の鍵を記録するEF2を設定する．さらにこのファイルでは，PIN照合と資格認証の両方が成功したときのみ読み出しができるようにするための論理記述ファイルを設定する．

以上のアクセス制限を付加すると，**図3**に示す保健医療カードの一部は，**図4**に示すファイル構造をもつことにより実現されることがわかる．この例では，救急情報や検診結果などに対する書き込み制限などについては触れていないが，現実には記録された情報の真正性を確保するためには，この制限や記録した人の電子署名を付加するなどの技術を加えることが不可欠である．また，ここでは詳しく述べていないが，EFに記録された鍵の照合は，単に鍵そのものを単純にカードに送って照合する手法から，端末とカードで共有している秘密鍵でアクセス鍵を暗合化して送る手法や公開鍵暗号方式を用いる手法まで，さまざまな手法が実用化されている．これらはカードがもつ機能と各アプリケーションの安全性に対する考え方により決まるものである．

上記の説明は主としてファイル構造に関するものであるが，各ファイルのツリー構造やアクセスコントロール用の鍵のID，さらにはその組み合わせを記録する論理構造などは，カード発行者が設定し，カードへアクセスする端末または人に，アクセス鍵を前もって渡しておくことが必要になる．また，カードに送られてくるコマンドを理解し，必要な処理を行うためのソフトウェアは，カード内のROMに記録するのが一般的である．

スマートカードの利用法

スマートカードの利用法は，重要な情報をカードに記録しオ

フラインで利用するデータキャリと，ネットワーク経由でデータベースなどにアクセスするための認証デバイスとしての使い方に大別される．

データキャリの使い方では，ネットワークを必要としないという利点がある反面，記録できるデータ量にはカードの記録容量の制限が存在する．これに対し，認証デバイスとしての使い方は，この逆の特性を有する．すなわち，後者ではカードに記録する情報は，ネットワーク経由で通信するホストコンピュータとの相互認証鍵が主であるため，カード内のメモリ容量はきわめて少なくて済む．もちろん，ネットワーク経由でアクセスする重要な情報は，ホストコンピュータ側で記録管理される．さらに，アクセス制御をホストコンピュータ側で管理することも可能であり，カードに記録しなければならない情報をさらに減らすことができる．

これら二つの利用法はそれぞれに利害得失があるため，一般的にはアプリケーションごとに両者を適切に組み合わせた利用法が用いられる．ただし今後は，ネットワークの普及と高度な暗号演算処理ができるカードの普及により，認証デバイスとしての利用法が増えると予想される．とくに，マルチアプリケーションカードの場合には，スマートカード自体が電子的な認証鍵の鍵束とみなせる認証デバイスとしての使い方のほうが，利用者への安心感の提供と柔軟性などからデータキャリ方式より優れているといえる．

近年スマートカードの導入が行われている主たる理由は，カードの偽造や変造を防止することとマルチアプリケーションなどの新たな利用法を実用化することに大別される．前者は，従来のテレホンカードやクレジットカードなどの置き換えとしてすでに実用化されている．この場合のICチップの役割は，券面などの偽造変造防止に加えて，チップ内に記録されるカード固有の鍵情報などを使ったカードそのものの正当性確認や，記録されている情報へのアクセス制限をかけることによる改ざん防止である．また後者は，住民基本台帳カードや市民カードなどとして開発/導入されようとしているもので，カード利用者が自由にサービスを選択するとともに，インターネット経由でさまざまなサービスを受けられるようにするものである．

IT社会と次世代スマートカード

ここでは，次世代スマートカードの必要性を明らかにするために，IT社会におけるスマートカードの役割を説明する．

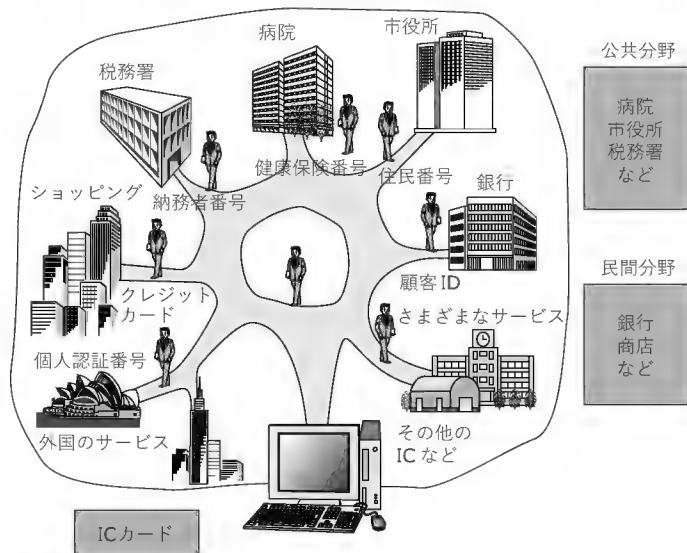
● IT社会とは

従来われわれは，現実空間において日々の社会活動を行っている．すなわち**図5**の概念図が示すように，物理的な空間において，たとえばショッピングセンターに行って買い物をする，役所に行って行政手続をする，あるいは病院に行って医師に見てもらうなどの社会活動をしている．そして，それぞれの場所でも現実空間において活動している．

これに対し，インターネットに代表されるオープンなネットワークにより「サイバー空間」と呼ばれる新たな場が作られ，この空間内にショッピングセンター，電子政府や医療機関などが構築されつつある(**図6**)．このサイバー空間内のショッピングセンターで買い物をするのが狭義のB to C(Business to Consumer)の電子商取引であり，同じくサイバー空間に開設された役所で行政手続をするのが電子政府の姿である．

もちろん，電子空間にわれわれが直接入ることはできないので，電子的な自分の代理を送らなければならないが，近未来には，サイバー空間でも現実空間と同じようにさまざまな社会活動が可能になると予想される．したがって，高度に情報化されたIT社会では，われわれの社会活動が従来の現実空間から電子的なサイバー空間にまで拡張すると予想される．そして，どちらの空間でどのような社会活動を行うかは，本人の意志にゆだねられるべきである．もちろん，われわれ皆がいつでも，どこでも，自由に，安全にサイバー空間においても，制度的および法

〔図5〕現実空間における従来の社会活動

〔図6〕サイバー空間における社会活動の概念図

的に有効な社会活動を可能にするためには，さまざまな問題を解決しなければならない．

たとえば，ショッピングに行くためには，現金あるいはクレジットカードなどの支払い手段が必要になるため，電子的なショッピングを実現するには，これらのものを電子化してサイバー空間に持ち込むことが必要である．また，電子的な申請/申告を可能にするためには，電子化された申請申告書を原本とする法的な措置などが必要になる．これらのことを一般化すると，われわれに帰属している現金などの有形物と，権利や義務などの無形物の両方を電子的にサイバー空間に持ち込むことが必要であることがわかる．とくに後者の無形物は，市民権と密接に関係するものである．

以上の説明からわかるように，IT社会の近未来像は，われわれの社会活動がサイバー空間にまで拡大し，現実空間と同じようにできるようになることである．

● スマートカードの役割

サイバー空間での社会活動を可能にするためには，前述したように，現金などの有形物と市民権などに関連する無形物の両方を電子化しなければならないが，ネットワーク化された社会では，これらの機能はサービス提供者側のサーバにより管理されるのが一般的になると考えられる．そして，提供されるサービスを利用する各人には，安全性を考えると数十から百バイト以上のディジタル情報が渡されると予想される．

この情報は，本人を特定するためのID情報とサービスを受けることの正統性を示す認証鍵の組み合わせなどになるが，個人情報保護の観点からこれらの情報はサービスごとに異なることが望まれる．そのため，利用者側から見ると自分が必要とするサービスの数だけ，このような桁数の情報を安全に管理しなければならないことになる．

スマートカードは，このように重要な情報を安全かつ確実に記録するための道具であり，その具体的な形の一つは，電子身分証明カード[1]という形態を取ると考えられる．したがってスマートカードは，いわば安全な電子財布であるといえる．そして，この財布に入れるものが前述のサービスを受けるために必要な情報であり，その種別は本人の選択にゆだねられるべきある．

また電子的な財布であるため，その数についても本人の自由になることが望まれる．さらに記録される情報は，前述したカードの利用法から見ると認証デバイスになるため，万が一紛失しても鍵の交換によりその安全性を十分確保することが可能である．

● 次世代スマートカードの必要性

IT社会の近未来像で示したように，誰もがサイバー空間に入って必要な社会活動ができるようにするためには，必要な情報が記録されているスマートカードは広域かつ多目的に利用できなければならない．なぜなら，自由に社会活動を行えるようにするためには，社会活動を行うために必要な情報を本人の希望により本人のカードに記録できることが必要であり，カードの能力と記録容量およびコストなどを考えると，複数のアプリケーションを一枚のカードがサポートする多目的カードが，そしてどこからでもサイバー空間に入れるようにするためには，カードの発行者や地域にしばられない広域利用ができるカードが必要だからである．

さらにスマートカードは，IT社会においてネットワークシステムに次ぐ第2のインフラとして期待されるが，インフラになるためには公共性および公益性が高く，さまざまアプリケーションで利用できることが必須である．具体的には上下水道の整備や電力の提供などと同じように，これらのインフラが整備されることにより新たな町ができ，さまざまなビジネスを生み出す効果を有することが必要である．

このことは言い換えると，既存のアプリケーションのみならず，今後期待される新たなアプリケーションにも対応可能なスマートカードが必要であることを示している．このような背景から，後述するように次世代スマートカードの開発が開始された．

NICSSの設立

次世代スマートカードシステムの開発普及を目標として，1998年12月に民間の任意団体として民間企業約70社からなる次世代ICカードシステム研究会（The Next generation IC Card System Study group．略してNICSS，ニックスと呼ばれている）が設立された．この研究会は，住民基本台帳カードをはじめとする公的分野での次世代スマートカードの実用化を図るために，現在の経済産業省，総務省，厚生労働省などとの意見交換を行い，次世代スマートカードシステムに対する要求定義をまとめている．さらに，マルチアプリケーションカードの運用/管理方式として「NICSSフレーム」を考案している．

NICSSによりまとめられた次世代カードシステムに対する要求定義の概要は，以下のとおりである．

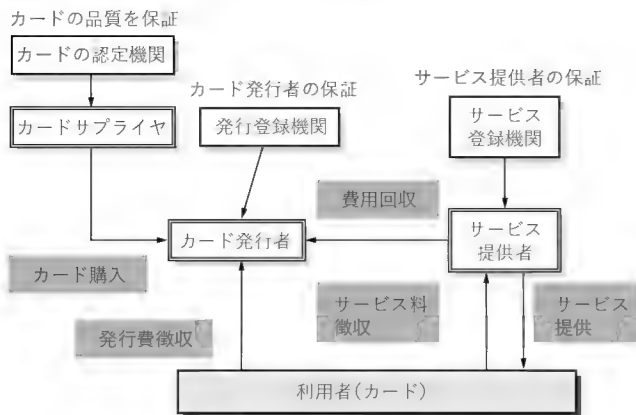
1) 広域/多目的なカードであること．とくに多目的をサポートするには，各アプリケーションは，ファイアウォールなどを用いて完全に独立すること
2) 電子署名に対応する高度な暗号演算機能を有していること
3) さまざまなアプリケーションに対応できる汎用性を有すること
4) 多種類のカード間の相互運用性が十分に確保されること

さらに，多目的カード利用の利便性とカードシステムの国際競争力を高めるために，

5) アプリケーションの追加/削除がカード発行元でなくサービス提供者のところでできること
6) 削除された領域の再利用が可能なこと
7) 近接型非接触インターフェースを有すること

が要求定義として追加されている．

また，上記の要求定義を満たすカードの試作は，（財）ニューメディア開発協会の「新カード研究開発事業」として平成10年，11年に行われ，その結果，JavaおよびC言語などを用いる次世

〔図7〕NICSSフレームのビジネスモデル

このモデルでは，カード発行者，カード利用者，サービス提供者の3者モデルになっている．また，それぞれの役割分担を明確し，十分な安全性を確保するために，必要に応じ認証組織の導入ができるようになっている．

代スマートカードが3種類試作された．この研究事業により次世代スマートカードの技術的な可能性は十分に確認され，後述する実証実験につながった．

NICSSフレームの考え方

前述した要求定義の5)は，次世代スマートカードシステムのもっとも大きな特徴の一つである．この要求定義を満たす汎用モデルとして，**図7**に示されるビジネスモデルがNICSSにより考案され，NICSSフレーム[2)]と呼ばれている．この方式の最大の特徴は，カード発行者，カード利用者，サービス提供者の3者モデルになっていることであり，従来のカードが用いている2者モデル(カード発行者とサービス提供者が一体になっている)と大きく異なっている．

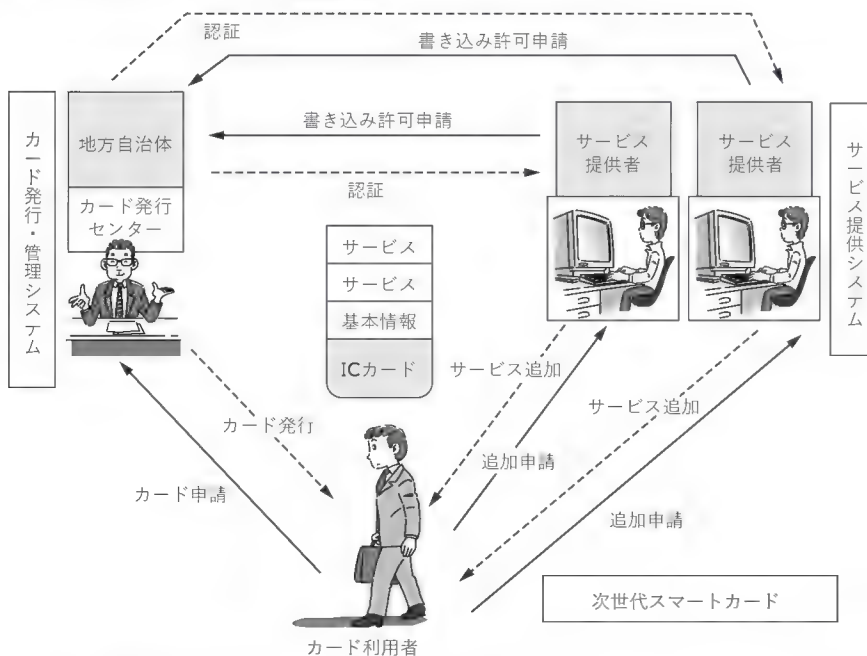
NICSSフレームにおけるカード発行者は，インフラとなるカードを発行するとともに，サービス提供者に対し，カードの一部の領域を無償ないしは安価に貸与する役を担う．こうすることでサービス提供者は，自らカードを発行するのに比べ，高機能かつ高い安全性を有するスマートカードを安価に利用することが可能になる．

さらに，カードに搭載するサービスをカード利用者が選択するという運用方式をとれば，結果として利用者が望むサービスのみが本人のカードに記録されるため，各利用者にとって，もっとも使い勝手の良いカードを作ることが可能になる．そして，アプリケーションの追加/削除は，上記要求定義の2)の機能を用いてカード発行者とサービス提供者間の通信をオンライン，オフラインを問わず安全に行うことができる．

NICSSフレームにしたがった次世代カードの運用例を**図8**に示す．この例では，はじめにカード利用者がカード発行者である自治体にカードの発行を申請する．すると自治体は，カード利用者の氏名などの基本情報を記録したカードを発行する．次にカード利用者は，自分の好きなサービスを選ぶ．従来のカードシステムではサービス提供者から新たなカードが発行されたが，このモデルではカード発行者にサービス追加の許可を求めることになる．そしてこの許可を得た上で，サービス提供者は当該サービスを提供するために必要となるソフトウェアのダウンロードあるいはDFを設定する．

ここで示した一連の手順を繰り返すことで，カードにはカード利用者が選んだ複数のアプリケーションが搭載される．このことからわかるように，カード利用者にとってもっとも使い勝手の良いカードができあがる．カード発行者にとっては利用者の満足感が得られるばかりか，カード利用者がサービスを選ぶたびに，カード費用の一部をサービス提供者から回収することができる．さらにサービス提供者にとっては，自らカードを発行する場合に必要となる発行システムなどの初期投資と運用管理コストを大幅に減じることができる．

〔図8〕次世代スマートカードの運用例

NICSSフレームは，従来のカードシステムにはない革新的なコンセプトである．そのため，欧米アジアなどの諸国から大きな関心が寄せられている．その例としては，EUが支援しているSCC(Smart Card Charter)と2001年，2002年の2年間にわたって技術協力を行い，さらに2003年以降も継続することがあげられる．また，VisaやMasterなどのクレジットカード会社が主メンバーとなるGP(Global Platform)とは，技術協力を進めるためのMOU(Memorandum of

Understanding)を締結し，積極的な交流を図っている．さらに，NICSSフレームの基本となるカードへのアプリケーションの追加/削除などは，「国際標準規格」の項で解説したように標準化されていない認証手順を必要とするため，現在，日米欧の協力の下で国際標準化の準備作業を行っている．

2階層のPKI

NICSSフレームを支える特徴的な技術は，2階層のPKIである．この考え方は，NICSSフレームが示すプレーヤモデルの責任分解から生まれたものである．すなわちNICSSフレームでは，従来のカードシステムと同様にカード発行者がカードを所有することから，カードの状態をコントロールする責任をもつとされている．具体的には，サービス提供者によるアプリケーションのダウンロードを可能にするために，当該カードをダウンロード可能状態にすることや，サービス提供者などの要望によりアプリケーションを削除することなどである．

このようにNICSSフレームでは，カード発行者は，カード利用者が選んだサービス提供者を知ることはあるが，カード内に記録されるソフトウェアやデータの内容を知ることはない．すなわち，これらのものはサービス提供者が責任をもつべきものであり，カード発行者が関与するものではないとしている．このような考え方は，サービスプロバイダセントリックと呼ばれている．

これに対し，従来のマルチアプリケーションカードでは，2階層のPKIを用いていないため，結果としてアプリケーションダウンロードの手法が公開されていないこともあり，カード発行者がダウンロードするイシュアセントリックな手法を用いている．後述する住基カードへの応用では，個人情報保護などの観点からも前者の手法が適していると考えられる．ただし技術的には，NICSSフレームの手法はどちらにも適用することができる．

次に，具体的にアプリケーションをダウンロードする手順について説明する．スマートカードは一般的に，電源が切れるとカードの状態がリセットされるため，次世代スマートカードでは通常，DFの創生やアプリケーションソフトのダウンロードに用いるシステムコマンドは受け付けないようになっている．このカードの状態を変えるには，カードがリーダ/ライタにセットされ電源が供給されている状態で，カード発行者との相互認証が成功することが必須である．そしてこの相互認証は，安全性とスケーラビリティの観点から非対称鍵暗号方式を用いており，カードに記録されているカード番号などを使うことで，カードごとに異なる鍵ペア(公開鍵と秘密鍵)の利用を可能にしている．

カード発行者との相互認証が成功すると，カードはダウンロードの許可モードに変わる．もちろん，ここでカードをリーダ/ライタから取り外すと，カードはリセットされダウンロードできなくなるので，この状態でサービス提供者は当該サービスが必要とするファイルあるいはソフトウェアをカードに書き込むことが必要である．さらに各サービス提供者は，独自のアプリケーションにおいて次世代カードがサポートする非対称鍵暗号を用いることができる．これらのことから，次世代スマートカードは2階層のPKIをサポートしていることがわかる．

IT装備都市研究事業

これまでの説明からわかるように，次世代スマートカードは従来のものから大きく発展しており，スマートカードがインフラとなるのに十分な機能を有している．このような可能性に着目し，2000年および2001年度には経済産業省の主導により「IT装備都市研究事業」が(財)ニューメディア開発協会[3]により実施された．この研究事業には，54の自治体と21の民間企業のコンソーシアムが参加し，公的分野および民間分野の各種のアプリケーションを実装した次世代スマートカードの大規模な実証実験が行われた．

用いたカードは4社の異なる製造メーカーから供給された5種類であり，それらはCPUやメモリサイズさらにはカードOSなどが異なっている．そして発行されたカードの総数は120万枚であり，現在も引き続き利用されている．

本事業で実証実験されたサービスは，①公共施設の予約，②医療保険証，③保健医療福祉介護への応用，④電子マネーなどの支払いサービス，⑤公共交通機関の電子チケットサービス，⑥商店街などのポイントサービス，⑦医師等の資格認証サービスなどである．これらのサービスは，各自治体から提案されたものであり，この実証実験により，提供されるサービスの任意の組み合わせが可能であることが明らかになった．

住民基本台帳カードへの応用

2003年8月には，2002年8月に実施された住民基本台帳改正法の規定により，希望する全国の住民に住民基本台帳カードが配布される予定である．この法律では，住民基本台帳に関連するアプリケーション以外のカードの空き領域は，市区町村の条例により他のアプリケーションを搭載してもよいことになっている．そして，2002年9月には住基カードの多目的利用を推奨するために，総務省から各自治体に対し条例の素案が提示されている．この素案には，前述したサービスを相乗りさせることなどが示されている．

一方，カードに記録される住基コードは，民間への利用が禁止されていることなどの理由により，カードにはきわめて高い安全性が要求されている．このような要求は，まさしく次世代スマートカードがめざしてきたものであり，前述の大規模実証試験の実績から，次世代スマートカードが採用されることが決まっている．さらに，採用されるカードの安全性については，第3者の専門家による評価/確認を行うこととしている．

住基カードは，次世代スマートカードが目標としてきたスマ

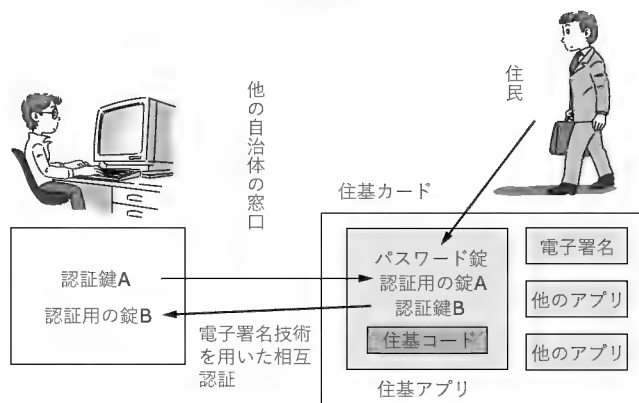
ートカードのインフラ化に大きく貢献すると期待されている．同時に，オープンで公平な市場を構築し，我が国のスマートカードシステム産業を育成する観点から，以下のような調達のプロセスが予定されている．

① 住基カードに対する要求定義の提示
② 供給希望メーカーとの秘密開示契約の締結
③ カードの安全性に関する第3者による評価と確認
④ 住基アプリ（住基カード本来がもつアプリケーション）の相互運用性の検証
⑤ 調達対象となるカードリストの自治体への提示
⑥ 調達対象リストに示されたカードの自治体による調達

2002年末の時点では，上記プロセスの②までが完了しており，③および④は2003年から開始する予定である．また⑤のリスト提示は2003年3月末を，⑥の調達は2003年4月から開始する予定である．

住基コードに対する読み出し許可のメカニズムを**図9**に示す．この図にあるように，カード利用者である住民は，自分のパスワード（4桁が想定されている）をキーインする．次に，端末とカードの相互認証が行われる．この相互認証は，2組のPKI（具体的にはRSAの1024ビット）が使われており，さらにカードに組み込まれている秘密鍵はカードごとに異なっている．これら三つの鍵の照合がすべて成功すると，端末は住基カード内の住基アプリに記録されている11桁の住基コードを読み出すことができる．このしくみは，技術的には一般的な電子署名で用いられるものと同程度の安全性を有しているので，住基コードの読み出しはきわめて難しく，実用上は十分と判断される．

図9で重要なもう一つの点は，公的個人認証サービスとして自治体から提供される電子署名の秘密鍵や証明書類も，追加される一つのアプリケーションとして住基カードに相乗りすることが想定されていることである．このことからも，住基カードはマルチアプリケーションに対応しなければならないことがわかる．さらに，用いるパスワードは，住基アプリと電子署名アプリではカード利用者の行為を明確に区別するために異なるものを設置できるように作られている．

〔図9〕住民基本台帳カードの内部構造の一例

おわりに

次世代スマートカードの本格的な実用化は，2003年8月から交付が予定されている住民基本台帳カードになると予想されるが，このカードシステム自体は決して公共分野の利用のみを考えて開発されたものではない．言い換えると，従来の少量多品種生産からインフラになりえる大量少品種生産にスマートカードを転換するために，もっともふさわしい主となるアプリケーションを探した結果，住民基本台帳カードをはじめとする公的分野のカードになったのである．もちろん，電子政府や電子自治体の構築が追い風になったことは間違いのない事実であるが，インターネットの開発と同じように開発された技術を民間に解放することで，次世代スマートカードが新たなインフラになることを強く希望する．

日本政府は今，電子政府や電子自治体を構築し，オンラインによる申請申告を可能にするために，官側ではGPKI（Government PKI．中央官庁の役職印の電子署名版）やLGPKI（Local Government PKI．地方自治体の役職印の電子署名版）を，民側に対しては法人認証サービスや公的個人認証サービスを実施あるいは準備中である．これらは，まさしくPKIの本格的な利用であるが，その規模は世界に類を見ないものである．

PKIが主としてソフトウェア技術により作られていること，利用者の利便性や安全性の観点からBCA（Bridge Certification Authority）を介して多数の認証局の相互承認が行われることなどを考えると，アルゴリズムや秘密鍵の危殆化（きたいか）に対しても被害を一部に封じ込める新しい技術の開発が重要である．そのためには，スマートカードとPKIを組み合わせることが有望と考えられるが，残念ながら，これら二つの重要技術を十分に理解している技術者はきわめて少ないため，早急に人材を育成し，IT分野における我が国の国際競争力を高めることが重要である．

住民基本台帳カードをはじめとする次世代スマートカードシステムは，我が国で開発された世界最高技術の一つである．そしてこの技術は2階層のPKIを用いていることから，インターネットを介したIP-VPNの鍵設定や情報家電などへのリモートアクセスを可能にすると期待されている．

これらの新規開発は，2003年から総務省および経済産業省の支援を得て開始される予定である．このような新しい技術を開発することは，IT社会がもたらす数々の利便性とメリットをわれわれ自身が享受するために，さらには我が国の繁栄・発展に大きく貢献するためにも必要である．

参考文献

1）大山永昭，「ICカードを用いた電子身分証明の構想」，『ITUジャーナル』，27巻9月号，（1997），pp.18-22
2）http://www.NICSS.gr.jp/
3）（財）ニューメディア開発協会のホームページ，http://www.nmda.or.jp/

おおやま・ながあき　東京工業大学　フロンティア創造共同研究センター

組み込みプログラミングノウハウ入門

第9回 時相論理とプログラム検証のはなし(その1)

藤倉俊幸

はじめに

連載の第7回「時間オートマタのはなし」(2002年12月号)のとき，時相論理式で時間オートマタの仕様を記述し検証できると説明した．このときは，Kronos[1]というツールを実際に使って状態マシン(ステートマシン)の合成を紹介した．そして，プログラム検証については，並行プログラムが仕様的にロックせずに動き続ける(none-Zeno性)かどうかを調べる方法のみを扱った．しかしKronosには，none-Zeno以外の並行プログラムの仕様をチェックする機能がある．今回は，このモデルをチェックする機能を使いこなすために必要な時相論理について準備をする．

Kronosは，CTL(Computational Tree Logic)という分岐時間時相論理(branching-time logic)に属する論理体系を使用しているが，とりあえず今回は時相論理の概要を説明するだけで，いろいろな体系については深入りしないことにする．

時相論理

● 様相結合子

時相論理で使用する□と◇は，様相結合子と呼ばれる(**図1**)．ある命題Pに対して，□Pはずっと Pが成立することを意味する．また◇Pは，いつかPが成立することを意味する．ここで命題とは，正しいか正しくないかが確定する文章や言明である．□とか◇が付かない命題は，今現在のことについて正しいか正しくないか言明していると考える．□とか◇が付いた命題は，将来のことについて現在の時点で語っている命題になる．たとえば，

◇□幸せ

は，「いつかずっと幸せになる」ということを主張する．様相結合子は，右側の命題に近いほうから結合する．この場合，まず(□幸せ)の部分が「幸せがずっと続く」ことを表し，それにさらに◇が付いた◇(□幸せ)は，「いつか幸せがずっと続くようになる」ことを表す．それでは，

□◇幸せ

は，どうなるだろうか．機械的に解釈すると「ずっといつか幸せ」ということになる．(◇幸せ)は「いつか幸せ」と読むと今は幸せでないと暗に意味していて，□(◇幸せ)はそれが永遠に続くという，全体として暗い意味になってしまう．実際はそのような意味ではなく，「ときどき幸せになったり，幸せでなくなったりすることがずっと続く」という意味である[注1]．

この場合，(□幸せ)の解釈は明確であるが，(◇幸せ)の解釈に曖昧さがあるようだ．自然言語に対応させてしまうと，このように意味が曖昧になってしまうので，時相論理では◇を以下のように定義する．

$$\Diamond P = \neg \Box \neg P$$

ここで，¬は否定を表す．つまり，◇Pは，「Pでないことがずっと続かない」という意味である．とすると，◇幸せは「不幸がずっと続かない」と解釈される．ただし，「いつか幸せ」と解釈すると，□幸せになってしまう．上記の式のほかに，

$$\neg \Box P = \Diamond \neg P$$

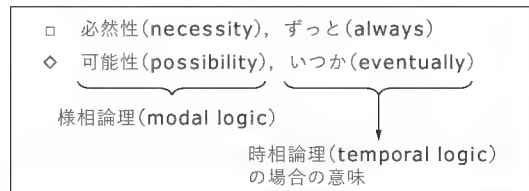
$$\Box \neg P = \neg \Diamond P$$

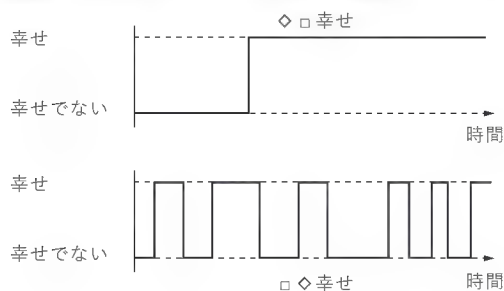
$$\Box P = \neg \Diamond \neg P$$

も成り立つので，時相論理式を自然言語によって解釈する際は，機械的に置き換えるのではなく，これらの関係を意識しながら解釈する必要がある．つまり，自然言語的には，◇Pか□Pの意味が明確なほうを基準にして，それを否定することで一方を解釈すると少しは理解しやすくなる．

論理式に□と◇を導入することで，**図2**で示した状態の変化を区別して表現できるわけだが，それだけなら単なる知的好奇心，悪くいえば言葉遊びとかわりない．組み込みエンジニアと

〔図1〕様相結合子の意味

□ 必然性(necessity)，ずっと(always)
◇ 可能性(possibility)，いつか(eventually)
様相論理(modal logic)
時相論理(temporal logic)の場合の意味

注1：無限回繰り返すことを◇∞と書く体系もあるが，一般には◇だけで何度も繰り返す意味になる．

〔図2〕「◇□幸せ」と「□◇幸せ」の違い

しては，だからどうなのかという次の展開が必要である．

● 強いセマフォ(Strongly-fair)と弱いセマフォ(Weakly-fair)

セマフォといえば，組み込みソフトでは，排他制御を実現するための定石というべきシステムオブジェクトである．このセマフォに，公平性に関して強いセマフォと弱いセマフォがある．市販のRTOSに実装されているセマフォはWait要求がキューイングされるので，いつか必ずブロック状態から開放される強いセマフォになっているので心配がない．しかし，自作のセマフォでポーリングベースの実装を行った場合，タイミングによって不公平な扱いを受ける可能性がある．

このようなセマフォを(公平性が)弱いセマフォと呼ぶ．同様な概念はセマフォ以外で，状態メッセージとイベントメッセージ，トリガ割り込みとレベル割り込みなどでも存在する．この時相論理的仕様を理解しないと，データの取りこぼしや状態の不整合などのバグの原因になる．これらは，実装仕様やインターフェース仕様が「◇□幸せ」か「□◇幸せ」のどちらになっているかに依存して決まってくる．こうなってくると，言葉遊びとは言っていられない．

セマフォSを簡単に定義すれば，S自身は整数値をとる変数で，Sを引き数とする二つの関数Wait(S)とSignal(S)により数値が変化する．

A）Wait(S)は，$S>0$であれば$S:=S-1$を実行し，$S<=0$のときはWaitを呼び出したタスクは実行を一時中断する

B）Signal(S)は，Sで実行を中断したタスクがあったら実行を再開する．なければ，$S:=S+1$を実行する

たとえば，セマフォをポーリングにより以下のように実装したとする．Wait(S)とSignal(S)は，別々のタスクから呼び出される．

```
void Wait(int S)
{
    for ( ; ; )
    {
        if(S > 0)
        {
            S = S - 1;
            return;
        }
    }
}

void Signal(int S)
{
    S = S + 1;
}
```

Wait(S)を呼び出したタスクは，$S>0$になるまで無限ループに入る．このような安易な実装をしたセマフォはまず存在しないと思うが，グローバル変数で同期をとるマルチタスクアプリケーションの設計やマルチプロセッサ間での同期をとるスピンロックの実装では，似たような状況になる可能性はあるかもしれない．たとえば，無限ループには入らないが，別の仕事をしながらときどきグローバル変数の状態を見にいくような実装である．

このWait(S)から抜ける条件は，◇□$(S>0)$になる．複数のタスクがWait(S)でループしている状況を考えてみると，$(S>0)$になったとたんに別のタスクが$S:=S-1$を実行して抜けてしまう可能性がある．すべてのタスクがWait(S)から確実に抜けるためには，□$(S>0)$がいつかは成立することが必要になる．このようなセマフォは，公平性が弱いセマフォである．

一方，強いセマフォでは，

```
void Wait(int S)
{
    if ( S > 0)
    {
        S = S - 1;
        return;
    } else
    {
        /* 待ち行列に入る */
    }
}

void Signal(int S)
{
    if (待ち行列が空)
    {
        S = S + 1;
        return;
    } else
    {
        /* 待ち行列から外す */
    }
}
```

のような実装になる．このWait(S)から抜ける条件は□◇(S>

0)になる．□(S > 0)になる必要はなくて，Wait(S)で待ち行列に入っているタスクの数だけ(S > 0)になるときがあれば十分である．したがって，◇(S > 0)がずっと保証されれば，いくつタスクがあってもすべてのタスクはいつかは待ちが解除される．

実際には，待ち行列がFIFOのような公平な待ち行列でない場合には，タスク数が3以上では要領の悪いタスクが待ち続けてしまうことが起こり得るが，とりあえず公平性の問題はセマフォの問題ではなく，待ち行列の問題に移されている．

セマフォの仕様を◇□(S > 0)か□◇(S > 0)で明確に示すことができる．これを自然言語で明確に示すには，冗長な表現になって読みづらくなる．一方，◇□(S > 0)と□◇(S > 0)を見分けるのは目が疲れるという意見もある．普通は，概念をおさえたら公平性が弱い，強いで表現する．さて，時相論理的な概念や仕様記述の重要性がわかったところで，組み込みエンジニアとしての次の展開に進もう．

2 プログラムの検証

要求仕様や設計仕様，実装仕様などを，論理式で表現すると何が良いのかという素朴な疑問がある．自然言語の曖昧さを排除できるということもあるが，それだけではなくプログラムの正しさを証明できるというところが大きなメリットである．このメリットを見るために，いったん時相論理からはなれて一般的なプログラムの話をする．

● テストしたところしかわからないのか？

開発したプログラムの正しさを調べるために，通常はテストを行う．たとえば，二つの変数xとyを使うプログラムのテストを完全に行うには，xとyのすべての組み合わせをテストする必要がある．変数の型が整数で32ビット環境であれば，$2^{32} \times 2^{32}$の組み合わせが必要になる．

しかし，このような膨大な数のテストをすることは不可能である．そこで，一つ一つの変数値の組み合わせを考えるのではなく，$x > 5$のような論理式でテストしてはどうかと考えてみる．そうすれば，テストの回数を減らすことが可能になる．プログラムの検証とは，データではなく論理式でプログラムのテストをすることだと考えるとよいかもしれない．

たとえば，与えられた整数xを2倍して1を加えるプログラムを作ったとする．プログラムは次のようになる．

$$y = 2 * x + 1$$

これを$\{ x = 3 \}$という条件のもとで実行すると，$\{(x = 3) \wedge (y = 7)\}$となる．これは普通のテストである．$\{ x \leqq 3 \}$という条件のもとで実行すると$\{(x \leqq 3) \wedge (y \leqq 7)\}$となるが，これは数学的に証明できる．この証明がプログラムの検証である．このことを，

$$\models \{ x = 3 \}\, y = 2 * x + 1 \{(x = 3) \wedge (y = 7)\}$$

とか，

$$\models \{ x \leqq 3 \}\, y = 2 * x + 1 \{(x \leqq 3) \wedge (y \leqq 7)\}$$

と書く．ここで，$\wedge$は論理積(and)を表し，$\models$は常に正しい(validである)ことを表す．Sをプログラムまたはその部分，pを事前条件，qを事後条件として，{ p } S { q }が常に正しいとは，pを満足する状態でプログラムSをスタートさせて，Sが終了したときqを満足する状態になっているということである．そして，実行の結果qがプログラムの仕様に合っていれば，プログラムの正しさが証明される．

プログラムの正しさを示そうとすると，事前条件はゆるいほど具合が良い．$\{ x = 3 \}$よりも$\{ x \leqq 3 \}$のほうがプログラムが正しいと主張(assertion)している範囲が広い．この事前条件が可能な入力範囲全体をカバーすれば，プログラムの検証が行えたことになる．プログラムSと事後条件qが決まれば，もっともゆるい事前条件も決まるので，もっともゆるい事前条件を，wp(S，q)と書く．wpはweakest preconditionの略である．

まず，簡単な例として，代入文についてwpを見てみよう．代入文については，

$$\mathrm{wp}(x := t,\ q(x)) = q(x)\ \{x \leftarrow t\}$$ 注2

が成り立つ．つまり，xにtを代入して事後条件$q(x)$となるようなもっともゆるい事前条件は，論理式$q(x)$の中のxをtで置き換えたものである．$\{x \leftarrow t\}$は，xをtで置換することを表す．代入文としての意味が逆のようだが，xにtを代入して$q(x) = \mathrm{True}$になる．そのためには，事前にxをtで置き換えた$q(x)$がpとして成立していればよいという当たり前の話である．

たとえば，

$$\mathrm{wp}(x := x - 1,\ x \geqq 0) = (x - 1 \geqq 0) = (x \geqq 1)$$

整数から1を引いてゼロ以上となるもっともゆるい条件は，最初に1以上であればよいわけだが，それは，事後条件の$x \geqq 0$のxを$x-1$で置き換えれば得られる．事前条件としては$x \geqq 2$とか$x \geqq 3$でも，事後条件$x \geqq 0$を満たすが，このような無数にある事前条件としてもっとも広い(弱い)条件が$x \geqq 1$である．

プログラム文が複数ある場合の事前条件については，

$$\mathrm{wp}(S1 : S2,\ q) = \mathrm{wp}(S1,\ \mathrm{wp}(S2,\ q))$$

のように，入れ子にして求めることができる．たとえば，

$x := x + a$;

$y := y - 1$;

を実行して，

$$x = (b - y) \times a$$

が成り立つためのもっともゆるい事前条件は，

$$\begin{aligned}
&\mathrm{wp}(x := x + a;\ y := y - 1,\ x = (b - y) \times a)\\
&= \mathrm{wp}(x := x + a,\ \mathrm{wp}(y := y - 1,\ x = (b - y) \times a))\\
&= \mathrm{wp}(x := x + a,\ x = (b - y + 1) \times a)\\
&= (x + a = (b - y + 1) \times a)\\
&= (x = (b - y) \times a)
\end{aligned}$$

となる．ここで面白いのは，得られた事前条件が事後条件と同

注2：論理式の「=」はC言語の「==」で，C言語の「=」つまり代入は，本文では「:=」によって表す．

一ということである．xとyの値はプログラムを実行することで変化しているが，xとyの間の関係は変化していないわけである．このような条件は不変条件と呼ばれ，プログラムの仕様の一部となっている場合もある．

`if`文については，

$wp(\text{if } B \text{ then } S1 \text{ else } S2,\ q)$
$= (B \rightarrow wp(S1,\ q)) \land (\neg B \rightarrow wp(S2,\ q))$

となる(**図3**)．`if`文はBが真であればS1を実行し，偽であればS2を実行する．S1かS2のどちらを実行しても，その結果qが成立する場合のもっともゆるい事前条件が，上で示したものになる．

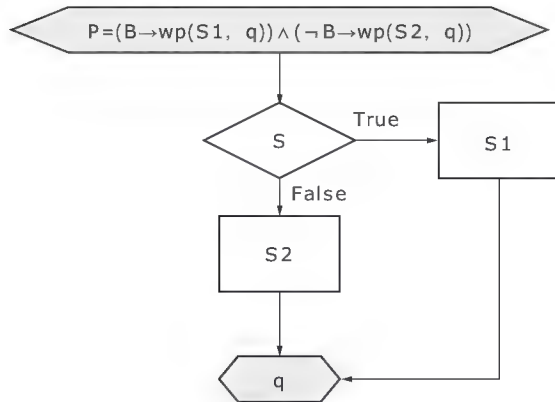
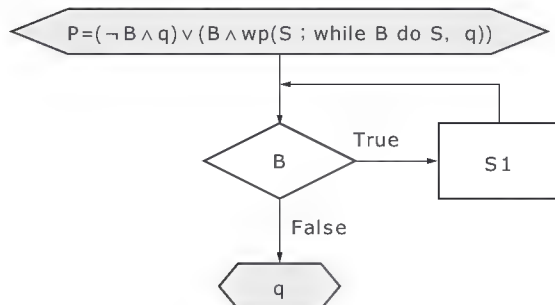
たとえば，

```
if ( y == 0 )
  x = 0;
else
  x = y - 1;
```

を実行して，

```
x = y
```

となるためのもっともゆるい事前条件は，

$wp(\text{if } y = 0 \text{ then } x := 0 \text{ else } x := y + 1,\ x = y)$
$= (y = 0 \rightarrow wp(x := 0,\ x = y)) \land (y \neq 0 \rightarrow wp(x := y+1,\ x = y))$
$= ((y = 0 \rightarrow y = 0) \land (y \neq 0 \rightarrow (y + 1 = y))$
$= \text{true} \land (y \neq 0 \rightarrow \text{false})$
$= \neg (y \neq 0)$
$= (y = 0)$

なので，$y = 0$になる．上記の式の変形の中で，$A \rightarrow B$でBがfalseであれば，$A \rightarrow B$がtrueになるのはAがfalseのときである，ということを使用した(**表1**のいちばん下の行参照)．

`while`文では，

$wp(\text{while } B \text{ do } S,\ q)$
$= (\neg B \rightarrow q) \land (B \rightarrow wp(S\ ;\ \text{while } B \text{ do } S,\ q))$
$= (\neg B \land q) \lor (B \land wp(S\ ;\ \text{while } B \text{ do } S,\ q))$

となる(**図4**)．`while`文はBが成立する間Sを繰り返し実行し，最終的にqが成立する．

Bが成立しなければ，ただちにqが成立する．その場合のもっともゆるい事前条件が，上の式から求まる．“ならば(→)”が論理式に入っていると扱いづらいので，上記の二番目の式は，

$(p \rightarrow q) \land (\neg p \rightarrow r) = (p \land q) \lor (\neg p \land r)$

を使って→を取り除いた．

例として，

$\text{while } (x > 0) \text{ do } x := x - 1\ ;$

で事後条件が$x = 0$になるもっともゆるい事前条件を考えてみる．簡単のために，上記の`while`文をWで表す．

$wp(W,\ x = 0)$
$= (\neg(x > 0) \land (x = 0)) \lor ((x > 0) \land wp(x := x - 1\ ;\ W,\ x = 0))$
$= (x = 0)) \lor ((x > 0) \land wp(x := x - 1\ ;\ wp(W,\ x = 0)))$
$= (x = 0)) \lor ((x > 0) \land wp(W,\ x = 0)\{x \leftarrow x - 1\})$

ここで，$wp(W,\ x = 0)\{x \leftarrow x - 1\}$を計算する必要が出てきたが，ここで$x$を$x - 1$に置き換えようとしている$wp(W,\ x = 0)$の中にもまた$wp(W,\ x = 0)\{x \leftarrow x - 1\}$が入っていることを考慮すると，

$= (x = 0)) \lor ((x > 0) \land ((x = 0)) \lor ((x > 0) \land wp(W,\ x = 0)\{x \leftarrow x - 1\}))\{x \leftarrow x - 1\}$
$= (x = 0)) \lor ((x > 0) \land ((x - 1 = 0)) \lor ((x - 1 > 0) \land wp(W,\ x = 0)\{x \leftarrow x - 1\}))\{x \leftarrow x - 1\}$
$= (x = 0)) \lor (x > 0) \land (x = 1) \lor ((x > 1) \land wp(W,\ x = 0)\{x \leftarrow x - 1\}\{x \leftarrow x - 1\})$
$= (x = 0)) \lor (x = 1) \lor ((x > 1) \land wp(W,\ x = 0)\{x \leftarrow x - 1\}\{x \leftarrow x - 1\})$

これを繰り返していくと，

$= (x = 0) \lor (x = 1) \lor (x = 2) \lor (x = 3) \lor \ldots$

つまり，

$= (x \geqq 0)$

となる．以上のことから，$x \geqq 0$が事前に成立していればwhile

〔図3〕IF文の場合

〔図4〕While文の場合

〔表1〕A→Bの真理表

A	B	A→B
T	T	T
T	F	F
F	T	T
F	F	T

〔図5〕並行プログラムの実行仕様(公理)

プログラム	実行仕様
l_i : v := 式	$l_i \to \Diamond l_{i+1}$
l_i : if B then l_t :　S1 　else l_f :　S2	$(l_i \land \Box B) \to \Diamond l_t$ $(l_i \land \Box \neg B) \to \Diamond l_f$
l_i : while B do l_t :　S1 ; l_f : S2	$(l_i \land \Box B) \to \Diamond l_t$ $(l_i \land \Box \neg B) \to \Diamond l_f$

$(x > 0)$ do　$x := x - 1$; を抜けた後 $x = 0$ が成立する，ということが証明された．

ここまでの話で検証できるのは，シングルスレッドでシーケンシャルな動作をするプログラムについてである．これらのプログラムはシーケンシャルなので，入り口と出口をおさえた，アサーションという形式{ p } S { q }で議論できる．プログラムSが停止した後に成立する条件 q を仕様としてプログラムの検証をしたり，p と q から S を自動合成したりできるわけである．

しかし，イベントを受けながら動き続ける組み込みシステムには終了ということがない場合もある．この場合，アサーション形式が使えない．また，マルチスレッドの並行プログラムの場合は，別のスレッドによって状況がいつの間にか変わってしまうこともある．この場合，論理式に□や◇を使うことになる．

3 組み込みプログラムの検証

まず，仕様表現から考えてみる．マルチスレッドが前提となる組み込みプログラムについて，論理式を使って仕様を記述する場合には，□と◇を使った時相論理式によらなければならない．たとえば，l_1，l_2，l_3 を文番号として if 文

```
l1 : if (B)
l2 :    S1 ;
      else
l3 :    S2 ;
```

があったときに，シーケンシャルプログラムであれば，

$$(l_1 \land B) \to l_2$$

$$(l_1 \land (B) \to l_3$$

のようにプログラムの仕様を記述できるが，並行プログラムの場合は，

$$(l_1 \land \Box B) \to \Diamond l_2$$

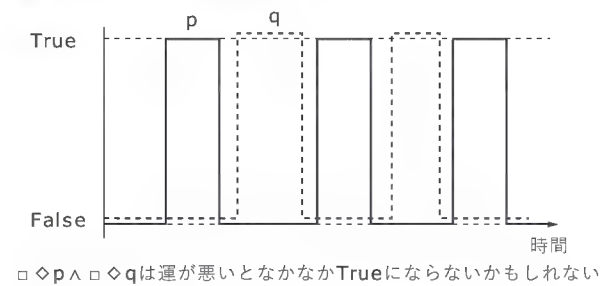
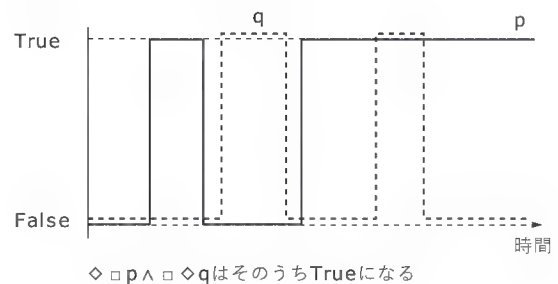
$$(l_1 \land \Box (B) \to \Diamond l_3$$

と書かなければならない．並行プログラムでは，あるスレッドの制御がある瞬間に l_1 にあっても，そのまま実行を続けられるとはかぎらない．別のスレッドに制御を取られてしまうかもしれない．その後，制御が戻って実行を再開したときに B の真偽が変更されているかもしれないので $(l_1 \land B)$ では不十分で，$(l_1 \land \Box B)$ としなければならない．

〔図6〕□◇p∧□◇qのグラフ

□◇p∧□◇qは運が悪いとなかなかTrueにならないかもしれない

〔図7〕◇□p∧□◇qのグラフ

◇□p∧□◇qはそのうちTrueになる

また，次の文が実行されるのはいつかそのうちなので，◇をつける必要がある．並行プログラムの実行仕様を**図5**にまとめた．**図5**の li や lt，lf は，文番号やプログラムカウンタ値，プログラムの状態などを表す．また，時間オートマタのロケーションに相当すると考えてもよい．

□◇p と◇□p 単体での違いについてはすでに述べたが，それらが組み合わさった□◇p∧□◇p と◇□p∧□◇p の違いにも注意する必要がある．組み込みプログラムのソースコードレビューやインスペクションを実施する際には，**図5**から**図7**までの実行仕様を頭に入れてコードを読まなければならない．

排他制御(mutual exclusion)の仕様は，たとえば同一のセマフォでプロテクトされたクリティカルセクション CS_1 と CS_2 があったときに，それらのクリティカルセクションに同時に二つのスレッドが入らない条件として，

$$\Box \neg (CS_1 \land CS_2)$$

のように表現される．しかし，この仕様は何もしないプログラムでも満足することができる．何もしないプログラムは CS_1 にも CS_2 にも入ることがないので，排他の条件を満足してしまうのである．デッドロックを起こして動けなくなっても排他条件を満たしてしまうということである．それで，排他制御を考えるときには，応答性(liveness)の条件も同時に考えなければならない．つまり，デッドロックしない条件である．応答性は，次のようになる．

$$\Box (Set_1 \to \Diamond CS_1) \land \Box (Set_2 \to \Diamond CS_2)$$

Set_1，Set_2 はクリティカルセクションの入り口の Wait(S)の部分である．Wait(S)していれば，いつかそのうちクリティカルセクションに入れる，ということが常に CS_1 と CS_2 について成

り立つ，という意味である．

図6の実行仕様は，基本的に同じ形をしている．つまり，ある文が実行されれば，いつかは次の文が実行されるという仕様である．$A \rightarrow \Diamond B$ の形を部分正当性（partial correctnesss）と呼ぶ．アサーション{ p } S { q }も，Sが終了するという条件が付いているので部分正当性に属する．終了すればとか，Aまで到達すればなどの条件を除いた条件を全体正当性（total correctness）と呼ぶ．

組み込みプログラムが満たすべき仕様を時相論理式で表現できたら，その式をプログラム情報を使って証明することが，そのプログラムの検証になる．

メモリへの書き込みがアトミックな処理であることを前提とした，二つのタスク間での排他制御アルゴリズムとして「デッカーのアルゴリズム[2]」がある．ソースコードで示すと**リスト1**のようになる．このアルゴリズムの応答性を例として検討してみたい．

P1とP2は別のスレッドのメイン関数として動作することを想定しているが，実際にはそれぞれ制御を離さないので，ラウンドロビンスケジューリングが使えない環境ではライブロックしてしまう．ここでは，P1の文とP2の文はアトミックであるがランダムな順番で実行されると仮定して，実際にどうやったら実現できるのかについては考えないことにする．ただし，このような仮定を設けると，一つのプログラムが動き続けてしまう状況が検証過程の中に混ざり込んでしまう欠点がある．公平性をどう仕様化するかについて，今回はふれないことにする．

〔リスト1〕デッカーのアルゴリズム

```
#define True 1
#define False 0

int C1 = False, C2 = False;
int Turn = 1;

void P1(void) /* Thread 1 */
{
    while(1) {
          /* non critical section */
          C1 = True;
          while(C2 == True) {          /* loop1 */
               if (Turn == 2) {
                    C1 = False;
                    while (Turn == 2); /* loop2 */
                    C1 = True;
               }
          }
          /* critical section */
          C1 = False;
          Turn = 2;
    }
}

void P2(void) /* Thread 2 */
{
    while(1) {
          /* non critical section */
          C2 = True;
          while(C1 == True) {
               if (Turn == 1) {
                    C2 = False;
                    while (Turn == 1);  <--P2待ち状態
                    C2 = True;
               }
          }
          /* critical section */
          C2 = False;
          Turn = 1;
    }
}
```

ここで考えたいのは，このようなプログラムは，どのようにテストすればよいのかということである．入力を与えて出力をチェックするテストはできないので，正しいことを証明するしかない．しかし，P1，P2は終了しないためアサーション型{ p } S { q }の証明はできない．よって別の方法によらなくてはならない．

また，C1，C2，Turnは別のスレッドによって書き換えられてしまうので普通の論理体系は使えない．そこで，時相論理式で仕様を表現してプログラムコードと対応させながら証明していくことになる．これはけっこう骨の折れる作業である．

まず，

$$\Box(C1 = True) \land \Box(Turn = 1) \rightarrow \Diamond \Box(C2 = False) \ldots (1)$$

を証明する．式の意味は，C1がtrue，Turnが1の状態がずっと続けば，最終的にはC2がFalseになるということである．プログラム的には，P1がクリティカルセクションに入ろうとすれば，いつかはP2が待ち状態になると解釈される．この式の前提部分を見るとC1がTrueに固定されているので，自由に動いているのはP2だけということになる．

すると，P2は非クリティカルセクションにいるか，クリティカルセクションまたはクリティカルセクションに入るためのプロトコルを実装した部分にいることになる．非クリティカルセクションにいるのであれば，C2 = Falseが成立する．プロトコル部分にいる場合はTurn = 1なので，C2 := Falseが実行されそのまま固定される．クリティカルセクションにいる場合は，最終的にクリティカルセクションを終了して，やはりC2 := Falseが実行される．クリティカルセクションは普通のプログラムなので，**図5**の実行公理が適用される．次に，

$$\Box(C1 = False) \land \Box(Turn = 2) \rightarrow \Diamond(Turn = 1) \ldots\ldots (2)$$

この式は，後で矛盾を導くために使用する．前提の□(Turn = 2)と結論の◇(Turn = 1)の部分が気になるが，論理式としての正しさは，前提が正しければ結論も正しいということだけではないので気にしないことにする（**表1**参照）．プログラム的意味は，この前提条件が成立すればP2はクリティカルセクションに入り，いずれ終了する，終了するとTurn:= 1が実行される，ということである．

以上の準備をしたところで，P1がプロトコル部分に入れば，最終的にクリティカルセクションに入れることを証明する．つまり，P1が無限待ちにならないことを証明する．証明は，背理法を使用する．つまり，無限待ちになることを仮定して矛盾を導く．

① $\Box\, Turn = 2 \rightarrow \Diamond \Box\, C1 = False$

仮定によって，P1はクリティカルセクションに入れないとすれば，C2 = Falseでloop1から脱出できないことになる．このとき，Turn = 2に固定されていればloop2からも脱出できないので最終的に，C1 = Falseに固定される．

② □ Turn = 2 → ◇ Turn = 1

式(2)と①から導かれる.

③ ¬ □ Turn = 2 → ◇ Turn = 1

もし，Turnが②でないとすればプログラム上，①にしかならない.

④ ◇ Turn = 1

②と③から導かれる.

⑤ ◇ □ Turn = 1

一度Turn = 1になると，P1がクリティカルセクションに入らないかぎり②にはならない．仮定により，P1はクリティカルセクションには入らないのでTurn = 1に固定される.

⑥ ◇ □ C1 = True

P1がTurn = 1でloop2から脱出すれば，C1はTrueになりそのまま固定される.

⑦ ◇ □ C2 = False

⑤と⑥と式(1)から導かれる.

*　　　　　*

いずれ，C2 = Falseに固定されるとすれば，P1はクリティカルセクションに入れることになり矛盾するので，仮定が否定されてP1の待ちが解除されることになる．これでP1の応答性が証明された.

結局，式(2)は□(C1 = False)と□(Turn = 2)が同時に真になることがないので前提部が偽となり，式(2)自身は真となる(**表1**参照)．プログラム的意味の，この前提条件が成立すればP2はクリティカルセクションに入り，いずれ終了する，終了するとTurn := 1が実行される，も間違いではないが，Turn = 1になればTurn = 2ではなくなるのとP1がクリティカルセクションに入る際にC1 := Trueを実行してしまうので□(C1 = False)と□(Turn = 2)が同時に真になることはないのである.

おわりに

今回は，プログラムの検証とはどんなものかを紹介して，それを組み込みプログラムで利用するには時相論理が必要になるという点にしぼって紹介した．実際に時相論理を理解して使いこなすには，さらに基本的なところから説明する必要がある.

次回は，再度時相論理について説明したいと思う．とくに，時相論理式の解釈の仕方や時間の概念の入れ方の説明が必要だろう．また，時相論理式と状態マシンとの関係が見えないと，実際の仕事でどう使うかがわかりにくいと思う．ということで，今回は参考文献3)から7)をあげて終わりとしたい.

参考文献

1) http://www-verimag.imag.fr/TEMPORISE/kronos/
2)『アルゴリズム辞典』，共立出版，1994
3) 田辺誠ほか，『論理とプログラム意味論』，岩波書店，1999
4) 萩谷昌己，『ソフトウェア科学のための論理学』，岩波書店，1994
5) M. Ben-Ari, *Principles of Concurrent and Distributed Programming*, Printice Hall, 1990
6) M. Ben-Ari, *Mathematical Logic for ComputerScience*, Springer, 2001
7) 山崎利治，「時間論理の解釈」，『デザインウェーブマガジン』，2002年5月号

ふじくら・としゆき　日本ラショナルソフトウェア(株)

高い信頼性が要求される
組み込み機器向けのRTOS

リアルタイムOS「INTEGRITY」の概要

森田 浩

「INTEGRITY」〔販売：(株)アドバンスドデータコントロールズ〕は，MMUによるメモリ保護，カーネルとユーザータスクの完全な分離により，安全性(secure)や可用性(availability)が最大限に重視される分野に適したリアルタイムOS(RTOS)です．

本稿では，最新の技術から生まれた「INTEGRITY」の，他の商用RTOSにはあまり見られない，保護とリソースの保証を提供する機能を解説します．

「INTEGRITY」の特徴的な機能は，次のようなものです．

- ハードウェアによるメモリ保護
- INTEGRITYカーネルとユーザータスクが完全に分離
- タスクは保護されたメモリ内で実行
- メモリ領域とCPU時間の保証
- 安全なタスク間通信
- Highest Lockerセマフォ
- 高速で時間予測可能なリアルタイム動作
- 極小の割り込み遅延時間
- 割り込みをいっさいマスクしないサービスコール

INTEGRITYカーネルは，オブジェクト指向カーネルです．

〔図1〕保護されたメモリ空間

プロセスまたはI/O装置のようなシステムリソースは，“Object”と呼ばれます．タスクは“Object”にメッセージを送ることで操作します．「INTEGRITY」には，次の9種類の“Object”があります．

- Semaphore
- Connection
- Clock
- Task
- AddressSpace
- MemoryRegion
- Activity
- Link
- IODevice

1. メモリ空間に関する保護

1.1 ハードウェアによるメモリ保護

現在，多くの高機能なプロセッサには，メモリマネージメントユニット(MMU)が内蔵されています．MMUの機能を利用すれば，不正なポインタアクセスなどによるプログラムの誤動作を発見することができます．

「INTEGRITY」では，MMUによるメモリ保護が採用されており，MMUにより保護された仮想アドレス空間を複数利用することが可能です(**図1**)．「INTEGRITY」とユーザータスクが完全に分離されるだけでなく，各タスクも保護されたメモリ空間で実行されます．その結果，あるタスクに不正な動作があったとしても，INTEGRITY自身や他のアドレス空間のタスクには悪影響が及ばないよう保護されます．

タスクが自身のメモリ空間を越えるアクセスをしようとした場合，MMUはそれをエラーとして例外を発生させます．カーネルはただちにそれを検出し，エラーの通知やログの作成をするとともに，当該タスクの削除や再スタートなどを行うことができます．

この不正メモリアクセスの検出は，とても重要です．不正アクセスがあった場合，ただちにその箇所が特定できます．しかし，メモリ保護機能がない場合では，メモリへの不正なアクセスがあってもただちにそれが検出できず，その後システムの異

常として検出されたところから不正アクセスの箇所を特定することはたいへんな作業になります．とくにタスクに割り当てられたスタックのオーバフロー，アンダフローは検出が困難ですが，MMUによってただちに検出できるので，エラーの発見をスムースに行うことが可能です．

● **AddressSpace Object**

アドレス空間(AddressSpace)は，それぞれが一つの論理的なメモリ空間で，データやプログラムコードが配置されます．複数のアドレス空間内には同じ論理アドレスが存在しますが，それらの実体は分離された物理アドレスに配置されます．すべてのタスクはいずれかのアドレス空間の中で実行されます．

INTEGRITYカーネルはそれ自身カーネル空間(Kernel Space)と呼ばれる一つの特別の物理的なアドレス空間の中で実行されます(**図2**)．仮想のアドレス空間は，コンピュータシステムの仮想記憶かメモリ保護ハードウェアによって実装されます．

〔図2〕INTEGRITYのアドレス空間

〔図3〕WeightによるCPU時間の割り当て

1.2 アクセス制御

ハードウェアによるメモリ保護だけでなく，タスクやその他のオブジェクトは，所有者やアクセス制限を生成時に静的に定義することができ，カーネルによるアクセス制御ができます．静的コンフィグレーションファイルにて定義された割り当ては固定的に行われ，後での変更/追加は不可能です．

2. メモリ領域とCPU時間の保証

他のタスク状態に関わらず，個々のアドレス空間内の各タスクに対して前もって設定したCPU時間やメモリ空間に関するシステムリソースを確実に提供し，利用させる機能です．

2.1 メモリ領域の保証

(1) カーネルでのメモリ利用保証

タスクがカーネルにどのようなサービスを要求しても，カーネルがその処理時にカーネルメモリ領域を使い切ってしまわないことを確実に保証する機能です．

この実現のために，サービス要求に応じて作成されるサービスメッセージやセマフォなどのカーネルオブジェクトにカーネルメモリをいっさい使用しない，という実装が必要で，「INTEGRITY」は，カーネル内部にはメモリプールをもちません．

(2) カーネル専用に分離されたスタック

カーネルが専用のスタックをもつことにより，ユーザータスクのスタックオーバフローを防止する機能です．カーネル専用スタックがない場合，カーネルはユーザータスクのスタックを使用します．つまり，ユーザータスクから見れば，未知のコードによりスタックが使われることになります．そのためユーザータスクが用意しなければならないスタックの最大サイズを決定することが非常に困難になり，スタックオーバフローのリスクが常に存在することになります．

2.2 CPU時間の保証

INTEGRITYでは，利用できるCPU時間がタスクとアドレス空間の両方に対して保証されます．重要なタスクやアドレス空間に(前もって割り当てた)必要なだけのCPU時間を，他のタスクやアドレス空間の状況に関わらず，常に提供できます．この機能により，他のRTOSで見られる「Denial of Service(DoS)」による問題を防止することができます．また，不適切なシステム設計やバグなどがあっても，重要なシステム要素の信頼性が維持できます．

● **Weight**

各タスクのCPU時間は標準のスケジューラで保証されます．例として，同じ優先度のタスクAとBがあるシステムで，タスクBがサブタスクB1とB2を生成する場合を考えてみましょう．INTEGRITYでは，タスクAとBに対して使用CPU時間比率〔INTEGRITYでは「重み(Weight)」と呼ぶ〕を設定し，それを保証することができます．

ここでは，それぞれ50%のCPU時間が前もって設定されているものとします．Bから生成される二つのタスクB1とB2に対しても同比率のCPU時間が設定されている場合，Bに割り当てられた50%のCPU時間が分割され，三つのタスクB，B1，B2に等しく割り当てられます(**図3**)．このように，タスクBでの動作がタスクAのCPU時間に影響を与えることはいっさいありません．

3. パーティションスケジューリング

INTEGRITYでは，オプションとしてARINC-653(後述)に準拠したパーティションスケジューラが用意されています(**図4**)．このスケジューラによりハードリアルタイムが実現されており，アドレス空間に対して前もって設定した(周期的な)タイミングと長さのCPU時間が，必ず提供されるようになります．

そのため，他のタスク状態に関わらず，そのアドレス空間内タスクの(周期的な)実行が保証されます．つまり，あるアドレス

空間にバグや悪意のあるコードが存在しても，あるいはウイルスやクラッカーが侵入しても，他のアドレス空間内のタスク実行に影響を与えることはありません．

4．安全なタスク間通信（コネクション）

MMUにより保護された仮想アドレス空間でタスクが動作しているとき，他のアドレス空間で動作するタスクとコミュニケーションをとる必要がある場合には，MMUのバリアを越えてデータを受け渡す機能がカーネルに必要になります．

● **Connection**

コネクション（Connection）はプロセス間，AddressSpace間およびINTEGRITY中のプロセッサ間コミュニケーション用のメカニズムです．タスクは，コネクションの一方の端の別のタスクにコネクションを通してメッセージを送信することができます．これは，相互プロセスコミュニケーションの理想的な手段です．それは安全で，異なるアドレス空間の中，あるいは異なるマシン上でさえ，タスク間でそうするのと同じくアドレス空間の中のタスク間で等しくうまくいきます．

データだけでなく，オブジェクトも別のタスクにコネクションを通して送ることができます．これは，異なるアドレス空間の中の関連するタスクがオブジェクトを共有することを可能にします．メッセージは，同期してあるいは非同期に送受信することができます．

5．Highest Lockerセマフォ

Highest Lockerセマフォ（優先度上限プロトコル）は，優先度逆転（**図5**）とチェーンブロッキングを防ぐことをめざしたバイナリセマフォの特別なものです（**図6**）．Highest Lockerセマフォには，生成時に静的な優先度（プライオリティ）が与えられます．タスクがHighest Lockerセマフォを使う場合，そのタスクの優先度がセマフォの優先度まで引き上げられます．

タスクがタイムスライスされない場合，セマフォの優先度はそのセマフォを要求するすべてのタスクの中のもっとも高い優先度とします．タスクがタイムスライスされる場合，セマフォの優先度はそのセマフォを要求するすべてのタスクの中のもっとも高い優先度以上のものとします．

6．リアルタイムパフォーマンス

INTEGRITYは，高速で時間予想可能なリアルタイム応答を実現できるように設計されています．システムコールのオーバヘッドを最小にするよう，すべてのカーネルサービスは注意深く最適化されています．処理時間の比較的長い複雑なシステムコールについては，処理途中でサスペンドさせ，他の仕事を先に完了させることもできます．

スケジューラはリアルタイムのスケジューラで，多レベルの優先度にしたがったスケジューリングを行います．さらに，同じ優先度レベルの各タスクに対しては，前もって設定されたCPU使用時間比率を保証したスケジューリングを行います．

INTEGRITYカーネルは，クリティカルなデータの操作中であっても，割り込みをマスクしません．それにより，最小の割り込み遅延を保証しています．またカーネルで使用する命令は注意深くチェックされ，パイプラインでの遅延が大きい命令（つまり，システムによっては長さが一定でない割り込みマスクを引き起こすような命令）の使用が回避されています．たとえば，除算命令やある種の文字列操作命令がこれに該当します．

INTEGRITYでは，最高優先度の割り込みは常に最小の遅延時間でサービスされます．実際に233MHzのPowerPCで遅延を測定をしてみると140ns程度です（**図7**）．同じシステムでのコンテキスト切り替えは870ns程度です（他の多くのRTOSより高速であることがわかる）．

7．開発環境とまとめ

これまで一般的なクロス開発ツールとして定評のあったGreen Hills Software社のMULTI統合開発環境が，INTEGRITYによるシステム開発にも対応しています．

MULTIとINTEGRITYを組み合わせた環境では，マルチタスクデバッグとして次のような機能を利用することができます．

● 複数CPU上での複数アドレス空間内にある複数のタスクを同

〔**図4**〕パーティションスケジューリング

〔図5〕チェーンブロッキングによる優先度逆転

〔図6〕Highest Locker(HL)セマフォによるスケジューリング

時にデバッグできる(個々のタスクには固有のデバッグウィンドウが割り当てられる)
- システム中のすべてのタスクが常にリスト表示される．そのリスト上で特定のタスクをクリックするだけで，そのタスクがデバッグ対象となる．そして，タスクリストウィンドウには，タスク名，状態，優先度，割り当てられたスタックサイズ，これまでの最大スタック使用量などがタスクごとに表示される
- タスクが生成されたときに，そのタスク用の新しいデバッグウィンドウを自動的に表示させることもできる
- 一つのタスクだけに有効なブレークポイントだけでなく，一つのアドレス空間に対して有効(そのアドレス空間内のすべてのタスクに対して有効)なブレークポイントも設定できる
- 多くのターゲットボードに対してマルチタスクデバッグがサポートされているが，ホストとの通信はシリアルポートまたはEthernet経由で行われる
- printfやscanfによるファイル/端末入出力を，ホストでの入出力でエミュレートできる
- (アイドルタスクを含めて)すべてのタスクの相対的な実行時間を表示することができる

*　　　　　　*

リアルタイムOS「INTEGRITY」はその特徴を生かし，航空，宇宙，ネットワーキング，テレコム市場で実績をあげています．

またINTEGRITYは，次のような国際規格に対応して開発されています．

- DO-178B(Level-A合致)

〔図7〕割り込み発生以降の処理の流れ

- ARINC-653 Application/Executive Interface(APEX)
- DO-255/RTCA SC-182 Requirements Specification for Avionics Computer Resource
- RTCA SC-182 Minimum Operational Standard for Avionics Computer Resource(MOPS for ACR)
- 共通基準 —— ISO 15408(オレンジブック)
- Ministry of Defense OO-55/OO-56
- Air Force Manual AFI 91-119 Nuclear Weapons System Software Certification
- Cenelec Railway Standard
- POSIX(1003-1b, 1i, 1c)

もりた・ひろし　(株)アドバンスドデータコントロールズ

宮坂電人

第1回 デメテルの法則

はじめに

本連載では，プログラミングをするにあたって要(かなめ)となる法則，方針，言い伝え，ノウハウを取り上げ，それらを解説します．

第1回は「デメテルの法則」と呼ばれている，プログラムをモジュールとして組み立てる際に必要になってくる，ある種のコツのようなものを紹介します．

デメテルの法則(The law of Demeter)

デメテルの法則で提唱されていることは，きわめてシンプルです．オブジェクト同士の相互関係を必要以上に複雑にしないため，以下のような方針を取るべきだということです．

- あるオブジェクトのメソッド内では，以下に示すオブジェクトのメソッドのみを呼ぶべきである．

① そのオブジェクト自身
② そのオブジェクトのメソッドのパラメータで指定したオブジェクト
③ そのオブジェクトが作成したオブジェクト
④ そのオブジェクトの直接のコンポーネント/オブジェクト

たったこれだけの簡単な話ですが，実際にこの法則を遵守することは簡単ではありません．というのも，あるオブジェクトが何かをしようとすれば，ここで示した以外のオブジェクトとも関係しなければならない状況になるので，どうしても法則を守れない場合が出るからです．ローカル変数と引き数だけでプログラムを構築するのが難しく，グローバル変数をアクセスしないといけない状況がありがちなのと似ています．

当然のことながらグローバルなものがからむと，それだけプログラムの挙動にバリエーションが出てくるためバグが発生しやすくなり，そのバグを取りにくくなります．同じことは，デメテルの法則を守らなかったプログラムでも生じてきます．

法則の出る背景

たいていのプログラムは，最初は単純なものから出発しますが，機能を追加するにつれどんどん複雑になっていきます．物理学のエントロピー増大則ではありませんが，プログラムの複雑度や乱雑度は増えることがあっても減ることはほとんどありません．

また，いったん増えてしまった複雑度/乱雑度を除去することは，プログラム全体を一から作り直すよりも難しい作業になりがちです．となると，われわれにできることは，複雑度/乱雑度の増大をなるべく少なくする工夫ができないかを検討することです．そのための策の一つが「デメテルの法則」です．

複雑度/乱雑度が増えるのはなぜなのでしょうか？　機能が増大するにつれ，量的にも質的にも“大きさ”が増えるからだろうと誰しもが直感的に思いつくところでしょう．機能の個数が増える，つまりコンポーネントやオブジェクトの個数が増えることで，

- コンポーネント/オブジェクト同士が結合する数が増えると，複雑度/乱雑度が増える
- 結合の組み合わせが増えると“組み合わせの爆発現象”により，複雑度/乱雑度が増える

といった“結合”に関しても個数や組み合わせが増え，それによって複雑度/乱雑度が増えるとも考えられます．ということは，

- 結合する数を減らすと，それだけ複雑度/乱雑度が減る
- 結合の組み合わせを減らすと，それだけ複雑度/乱雑度が減る

という考えにいきつきます．

ただし，どういう基準で数/組み合わせを減らすかが曖昧では，手当たりしだいに減らそうとして，本当は削るべきでなかったものを削ったり，あるいはその逆の削っていいものを削っていなかったことによる支障が出るおそれがあります．

そのような事態に陥らないためには，何らかの判断基準が必要です．このような場合にデメテルの法則を適用します．つまり，

- 自分や自分の身内，自分にとって“密接なもの”は必要だから削らない
- 自分や自分の身内からへだたっている“密接でないもの”は削る

という“自分からの密接度”を基準にして，結合する/しないを決めることで，結合する数/組み合わせを減らし，そのことで複雑度/乱雑度を減らすことができるわけです．

また，自分からの距離や密接度の判断基準として，次の四つを利用すればよいわけです．

① そのオブジェクト自身：自分自身であり密接度が最高だと判断
② パラメータで指定したオブジェクト：パラメータとして指定されたことは，その時点で密接度が高いと判断
③ 作成したオブジェクト：必要があって作成したわけだから密接度が高いと判断
④ 直接のコンポーネント/オブジェクト：必要があって直接の関係をもつわけだから密接度が高いと判断

見知らぬ人とは話すな

デメテルの法則と似たもので，「Edelman's law」というのがあります．ただし，こちらはデメテルほど有名ではなく，ほとんど知られていないと思われます．簡単にいえば，"Don't talk to strangers"(見知らぬ人とは話すな)ということで，やはり密接な関係ではないオブジェクトやコンポーネントと直接かかわらないように，という戒めです．

しかしこれも，グローバル変数なしでプログラムを組むのと似ていて，できなくもないけれど，やろうとするととたんに難しいと感じてしまいます．

デザインパターンとのからみ

昨今はやりつつあるデザインパターンでいえば，デメテルの法則の応用ともとれるパターンは，

- Mediatorパターン：Mediatorと呼ぶ仲介者を利用することで特定のオブジェクト同士の結合をなくす

だと考えられます．あるいは，

- Facadeパターン：窓口となるオブジェクトを決めることでサブシステムの取り扱いを単純化する

あたりでしょうか．いずれのパターンも，複雑度/乱雑度をなるべく減らそうという意図があります．詳細に検討していくとほかのデザインパターンでもデメテルの法則がからむものがありそうですが，それだけ同法則が普遍的に適用できる可能性があるとも考えられます．

サンプルプログラム

実際に，プログラムでデメテルの法則を利用するとはどういうことなのかを説明するため，簡単なサンプルプログラムを示しましょう．次のような命題のプログラムを考えます．

- あらかじめ決めた営業時間のみモータを動作させるプログラムを作る
- 温度センサを監視して，モータの保証温度範囲外なら停止させる
- モータの動作をログにして残す

検討した結果，次のようなオブジェクトが見い出せたとします．

- モータ駆動オブジェクト：モータのON/OFFを制御する
- 営業時間オブジェクト：営業時間であるかないかを判断する
- 温度センサオブジェクト：温度センサから温度情報を得る
- ログ記録オブジェクト：ログを記録する

それぞれのオブジェクトを実現するクラスは，**リスト1～リスト5**のようなJavaプログラムで記述できたとします．

〔リスト1〕モータ駆動クラス

```
public class motor driver {
    public motor_driver(){  //コンストラクタ，モータの初期化など
        ...(省略)...
    }

    public void on(){        //モータをONにする
        ...(省略)...
    }

    public void off(){       //モータをOFFにする
        ...(省略)...
    }
}
```

〔リスト2〕notify_current

```
public interface notify_current {
                              //範囲内か範囲外かを知らせるメソッドをもつインターフェース
    final int notify current over = 1;    //範囲の上限を越えている
    final int notify_current normal = 0;  //範囲内にある
    final int notify current under = -1;  //範囲の下限を越えている
    public int notify_current status();
                                          //範囲内か範囲外かを知らせるメソッド
}
```

〔リスト3〕営業時間クラス

```
public class business hour implements notify current {
    public business hour(){ //コンストラクタ，内部状態の初期化など
        ...(省略)...
    }

    public int notify current status(){ //営業時間内かどうかをえる
        ...(省略)...
    }
}
```

〔リスト4〕温度センサクラス

```
public class temperature implements notify current {
    public temperature(){ //コンストラクタ，内部状態の初期化など
        ...(省略)...
    }

    public int notify current status(){ //許容温度内かどうかをえる
        ...(省略)...
    }
}
```

〔リスト5〕ログ記録クラス

```
public class log recorder {
    public log recorder(){ //コンストラクタ，内部状態の初期化など
        ...(省略)...
    }

    public void record(boolean iMotorOn){
                                      //モータのON/OFF状況を記録する
        ...(省略)...
    }
}
```

〔リスト6〕メイン部分(1)

```
static void bad_sample(){
    new bad_motor_task();                               //モータ駆動担当スレッドを起動する
    new bad_temper_task();                              //温度センサ担当スレッドを起動する
    new bad_hour_task();                                //営業時間担当スレッドを起動する
    for(boolean aContinue = true; aContinue; ){
        ...(省略)...
    }
    bad_motor_task.instance.end_task();                 //モータ駆動担当スレッドを終了する
    bad_temper_task.instance.end_task();                //温度センサ担当スレッドを終了する
    bad_hour_task.instance.end_task();                  //営業時間担当スレッドを終了する
}
```

〔リスト7〕モータ駆動担当スレッド(1)

```
public class bad_motor_task implements Runnable {
    public static bad_motor_task instance;              //唯一のインスタンスを指す変数
    private log_recorder recorder;                      //ログ記録オブジェクトを指す
    private motor_driver motor;                         //モータ駆動オブジェクトを指す
    private boolean continue_flag;                      //trueである限り,スレッドは継続する
    private boolean motor_status;                       //モータの現状のON/OFF状態を示す

    public bad_motor_task(){                            //コンストラクタ,ここでスレッド起動も行う
        instance = this;                                //自身を唯一のインスタンスとする
        continue_flag = true;                           //スレッドを継続する
        recorder = new log_recorder();                  //ログ記録オブジェクトを作成する
        motor = new motor_driver();                     //モータ駆動オブジェクトを作成する
        motor.off();                                    //モータをOFFにする
        motor_status = false;                           // "
        Thread aThread = new Thread(this);              //スレッドを起動する
        aThread.start();                                // "
    }

    public void end_task(){                             //スレッドを終了する
        continue_flag = false;                          //継続フラグを落とす
    }

    public void run(){
        while(continue_flag){
            ...(省略)...
        }
    }

    public synchronized void change(boolean iOn){       //モータのON/OFFを指示する
        if(iOn != motor_status){                        //現状のモータの状態から変化するなら
            if(iOn){                                    //モータのON/OFFをする
                motor.on();
            }else{
                motor.off();
            }
            recorder.record(iOn);                       //ログを記録する
            motor_status = iOn;                         //状態フラグを変える
        }
    }
}
```

〔リスト8〕営業時間担当スレッド(1)

```
public class bad_hour_task implements Runnable {
    public static bad_hour_task instance;               //唯一のインスタンスを指す変数
    private boolean continue_flag;

    public bad_hour_task(){
        instance = this;
        continue_flag = true;
        Thread aThread = new Thread(this);
        aThread.start();
    }

    public void end_task(){
        continue_flag = false;
    }

    public void run(){
        business_hour aHour = new business_hour();      //営業時間オブジェクトを発生する
        while(continue_flag){                           //スレッドが続く限り,営業時間内ならモータON,時間外ならモータOFFにする
            bad_motor_task.instance.change(aHour.notify_current_status() == business_hour.notify_current_normal);
            ...(省略)...
        }
    }
}
```

〔リスト9〕温度センサ担当スレッド(1)

```
public class bad_temper_task implements Runnable {
    public static bad_temper_task instance;                //唯一のインスタンスを指す変数
    private boolean continue_flag;

    public bad_temper_task(){
        instance = this;
        continue_flag = true;
        Thread aThread = new Thread(this);
        aThread.start();
    }

    public void end_task(){
        continue_flag = false;
    }

    public void run(){
        temperature aTmp = new temperature();              //温度センサオブジェクトを発生する
        while(continue_flag){                              //スレッドが続く限り，許容温度内ならモータON，外ならモータOFFにする
            bad_motor_task.instance.change(aTmp.notify_current_status() == temperature.notify_current_normal);
            ...(省略)...
        }
    }
}
```

いずれも単機能であり，このままでは目的のプログラムには，ほど遠いものです．そこで，それぞれの単機能なクラスを利用するスレッドを複数作っていくことにします．

デメテルの法則を無視した例

最初に示すのは，筆者がよく見かける「手当たりしだいにプログラミングした」プログラムです．あらかじめお断りしますが，このプログラムには致命的な間違いが含まれています．それがどういうものであるかも探しながら見ていただくと，よりいっそう，プログラムのもつ問題点もはっきりすると思います．

まずはメイン部分です．三つのスレッドを起動し，最後にすべてのスレッドを終了させるという単純なプログラムです(**リスト6～リスト9**)．

ここで気付くことは，各スレッド同上の連係です．それぞれのスレッドにinstanceという公開変数をもたせて，これを介して，それぞれのスレッド同上でメソッドの呼び合いをしています．

この方法をとっていることに何か問題があるのか，と疑問に思う人もいるでしょう．よく考えていただきたいのは，それぞれのスレッド同上の関係が平等で平板に見えるため，どういう情報伝達がなされているのか曖昧になる危険をかかえているということです．このプログラムでは，

- 営業時間担当スレッドがモータ駆動担当スレッドのchangeメソッドを呼び出す
- 温度センサ担当スレッドがモータ駆動担当スレッドのchangeメソッドを呼び出す

という安直なことをしています．気付いた人もおられるでしょうが，「営業時間内→営業時間外でモータOFF」となった後で「温度範囲内→温度範囲外→温度範囲内でモータON」になるような事象もあり得ます．この場合，営業時間外なのにモータがONになってしまって都合が悪いわけです．

平板で平等な関係でモジュールを構成した場合，このような，あちらこちらからアクセスが起きる不都合な事態をまねく可能性が増え，そのために不都合な事態をおさえる工夫が必要になるのですが，たいていは後付けの“パッチ”となりプログラムが汚くなったり，仕様追加が発生すると，再び後付けのパッチに悩むハメになります．

デメテルの法則を守った例

デメテルの法則を守った場合，まずinstanceのような公開変数は極力避けるべきです．というのも，法則のすすめる“密接なもの”同上で連係するという構造は，公開変数を利用すると崩れ去ってしまうからです．したがって，さきほどのプログラムをなるべくいじらないように，ただしデメテルの法則を守って作成しなおしましょう．

リスト10はメイン部分です．さきほどと違い，モータ駆動担当スレッドのみです．じつは，ほかのスレッドはモータ駆動担当スレッド内に抱きかかえています．つまり，デメテルの法則でいうところの「③そのオブジェクトが作成したオブジェクト」にしてしまうわけです(**リスト11**)．

ただし問題は，作成された側(ここではgood_hour_taskオブジェクトとgood_temper_taskオブジェクト)から，作成した側(good_motor_taskオブジェクト)へ情報を伝達したい場合はどうするかです．

作成した側を公開変数にすると，せっかく意図したことが崩れてしまうので，この場合，やはり法則の「②そのオブジェクトのメソッドのパラメータで指定したオブジェクト」としてしまうわけです．

ここでは具体的には，作成される側のコンストラクタのパラメータとして，作成した側を引き渡します．引き渡された側(つ

〔リスト10〕メイン部分(2)

```
    static void good sample(){
        good_motor_task aMotorTask = new good_motor_task();   //モータ駆動担当スレッドを起動する
        for(boolean aContinue = true; aContinue; ){
            ...(省略)...
        }
        aMotorTask.end task();                                //モータ駆動担当スレッドを終了する
    }
```

〔リスト11〕モータ駆動担当スレッド(2)

```
public class good motor task implements Runnable {
    private log_recorder recorder;                       //ログ記録オブジェクトを指す
    private motor driver motor;                          //モータ駆動オブジェクトを指す
    private boolean continue flag;                       //trueである限り,スレッドは継続する
    private good_hour task hour task = null;             //営業時間担当スレッドを指す
    private good temper task temper task = null;         //温度センサ担当スレッドを指す
    private boolean motor_status;                        //モータの現状のON/OFF状態を示す

    public good motor task(){
        continue_flag = true;                            //スレッドを継続する
        recorder = new log recorder();                   //ログ記録オブジェクトを作成する
        motor = new motor_driver();                      //モータ駆動オブジェクトを作成する
        motor.off();                                     //モータをOFFにする
        motor status = false;                            // "
        Thread aThread = new Thread(this);               //スレッドを起動する
        aThread.start();                                 // "
    }

    public void end_task(){
        continue_flag = false;
    }

    public boolean is continue(){                        //スレッドが継続しているかを見る
        return continue flag;
    }

    public void run(){
         hour task = new good hour task(this);           //営業時間担当スレッドを作成する
        temper task = new good temper task(this);        //温度センサ担当スレッドを作成する
        while(continue flag){
            ...(省略)...
        }
    }

    public synchronized void update(){                   //モータのON/OFF状況を更新する
        boolean a new status = (hour task != null && hour task.is ok() && temper task != null && temper task.is ok());
        if(a new status != motor status){
            if(a new status){
                motor.on();
            }else{
                motor.off();
            }
            recorder.record(a new status);
            motor status = a new status;
        }
    }
}
```

まり作成される側)では，この情報を法則の「④そのオブジェクトの直接のコンポーネント・オブジェクト」つまりインスタンス変数として保持すればよいわけです．

ところで一箇所，意図を説明しておかないとわかりにくいメソッドがあります．さきほどは，モータのON/OFFをchangeメソッドで実装したのですが，ここではupdateメソッドに変更しています．changeメソッドでは単純にモータのON/OFF切り換えをしているだけで，それが“全般を把握した挙動”にならない悩みがありました．かりにここで，湿度センサを追加して許容湿度内のみモータをONにするという追加仕様があったとすれば，とたんに頭をかかえることになります．

物の見方を変え，外部から告知するのはモータのON/OFFの直接指示ではなく，モータのON/OFF変更の要請と考えてみるわけです．モータをONにするかOFFにするかの主導権はモータ駆動スレッドが握り，どちらに傾けるべきかは営業時間がどうであるか，温度センサがどうであるかをモータ駆動スレッドから働きかけて調べるというスタイルになります．こうしておくと追加仕様があった場合，追加は調べるべき対象の追加という楽な方向ですみます．

デメテルの法則を適用した場合，プログラムの追加やデバッグは，このようなモジュールの密着した切り口を経由して伝搬するというスタイルを取りやすいため，その分楽になるわけで

〔リスト12〕営業時間担当スレッド(2)

```
public class good_hour_task implements Runnable {

    private good_motor_task motor_task;
    private business_hour mHour;

    public good_hour_task(good_motor_task iTask){
        motor_task = iTask;
        mHour = new business_hour();
        Thread aThread = new Thread(this);
        aThread.start();
    }

    public void run(){
         while(motor_task.is_continue()){
            ...(省略)...
            motor_task.update();                    //モータのON/OFF状況の変化を告知する
            ...(省略)...
        }
    }

    public boolean is_ok(){                         //モータをONにすべきか(true), OFFにすべきか(false)をえる
        return (mHour.notify_current_status() == business_hour.notify_current_normal);
    }
}
```

〔リスト13〕温度センサ担当スレッド(2)

```
public class good_temper_task implements Runnable {

    private good_motor_task motor_task;
    private temperature mTmp;

    public good_temper_task(good_motor_task iTask){
        motor_task = iTask;
        mTmp = new temperature();
        Thread aThread = new Thread(this);
        aThread.start();
    }

    public void run(){
        while(motor_task.is_continue()){
            ...(省略)...
            motor_task.update();                    //モータのON/OFF状況の変化を告知する
            ...(省略)...
        }
    }

    public boolean is_ok(){
        return (mTmp.notify_current_status() == temperature.notify_current_normal);
    }
}
```

す．法則を無視すると，どこから攻めるべきかを調べる手間が発生しやすくなります．

「モジュールの密着した切り口を経由して伝搬」という特徴によって起きるプログラムのスタイルの変化として，

- 同じ意味のメソッドや変数を統一しやすくなり，無駄な記述が減る

という特徴があります．ここでは，具体的にはis_continueメソッドがあげられます．先ほどの例では，それぞれのスレッドで継続フラグ(continue_flag)や継続停止メソッド(end_task())を記述していましたが，スレッドの停止判断(is_continue())がメインとなるスレッドに集中したことで，同じような記述が減らせる場合があります(**リスト12**，**リスト13**)．

しかし，これとは逆の現象——

- 密着していないモジュールのメソッドを利用するため，カスケードが起こりやすくなる

という特徴もあります．たとえば，A→B→Cという密着をしている場合，AからはCにあるメソッドを直接利用できないので，Bに“中継メソッド”を用意するような現象が起きる場合があります．ただし，上手に運営することで，法則に遵守したまま中継メソッドを減らしたり，なくしたりすることは可能です．

みやさか・でんと　miyadent@anet.ne.jp

Ogg Vorbisの技術とオープンオーディオライセンス

第2回

岸 哲夫

今回も前回に引き続いて，OggVorbisについての話題を続けます．

前回は，Vorbisが公式にサポートしているツール類を紹介しましたが，他にもVorbisに対応しているプレーヤが，プラットホームごとにいくつか存在します．

`http://www.vorbis.com/software.psp`

に記述がありますが，BeOS，Java，MacOS，Linux，Linux Development，Mac OS X，OS/2，Win32 Development，Windowsの各プラットホーム別にリンクが張られています．中にはJavaで作られているものもあるので，PDAでも利用できます．

Vorbisのデコードはプロセッサにとって荷が重い処理のようで，まだ携帯用ディジタルオーディオプレーヤでVorbisに対応した製品は発表されていません．

アイリバー社(**図1**)の製品であるiMP-350は，ファームウェアのアップグレードによってVorbisフォーマットに対応するとしています．近いうちにVorbisフォーマット対応ファームウェアが発表されることでしょう．

〔図1〕アイリバージャパンのWebページ
(`http://www.iriverjapan.com/`)

Vorbisのチャネルカップリングと特別なステレオ音響処理について

VorbisオーディオCODECは，チャネル間の余剰の除去およびステレオイメージ情報を除去することによってビットレートを縮小します．そして，音響心理学の理論により，聞き取れない音や不適当な音は，xiph.org(`http://www.xiph.org/`)によって提供されるlibvorbis CODECによって機械的に処理されます．

エンコーダリリースのベータ4以前では，Vorbisはmultiple channel encodingに対応していました．しかし，チャネルは，まったく別々にコード化されるとは限りませんでした．その場合の問題点は，チャネル間のクロス分析と余剰の除去でした．

従前の多重チャネルの戦略は，MP3の2重のステレオモードに非常に似ています．現在の仕様ではチャネルを分離し，連結する方法をとっています．

Vorbisは，チャネルカップリングをインプリメントするための二つの技術をもっています．

一つはチャネルインタリービングであり，そしてもう一つがsquare polar mappingという手法です．これによりチャネルの分離，および正確なエンコードを実現しています．

● square polar mappingについて

チャネル間の相関性について説明します．

Vorbisの基本的処理として，入力ストリーミングからMDCT-domain whitening filterを通してスペクトルのfloor functionを作り出します．このfloorは，周波数スペクトルの大まかなエンベロープを表します．

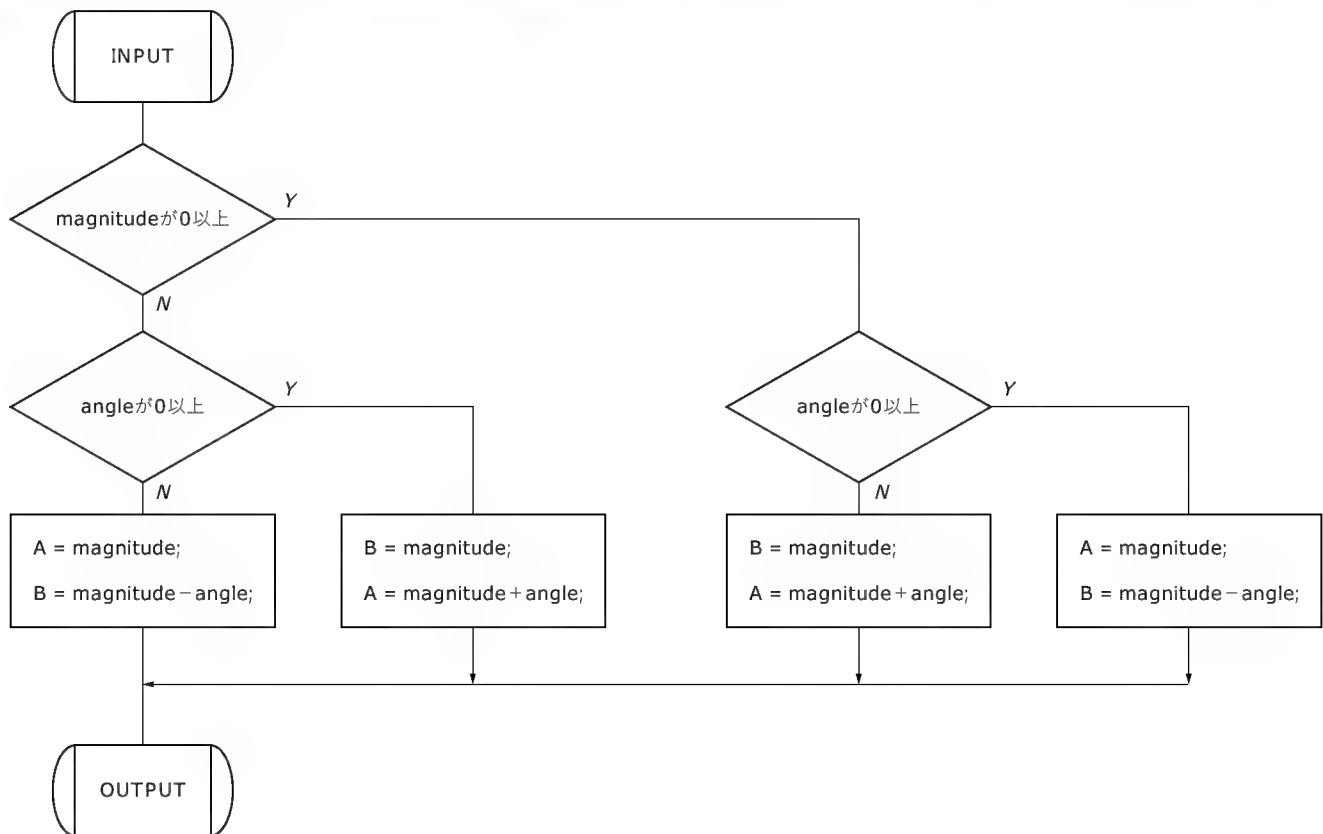
このfloorは，周波数によって有効にスペクトルをノーマライズし，周波数スペクトルから演算されます．各入力チャネルは独立したfloor functionをもっています．

ステレオカップリングの基礎的な考え方は，左と右のチャネルが通常は相互に関連するということです．

floor functionは，カップリングの前に相関性を最大限にするため，チャネルを横断してスペクトルを標準化します．これは決してVorbisフォーマットのバグではないようです．

最大に左と右のチャネルを関連させて，効率的に左と右のチ

〔図2〕Square Polar Mappingにおいて情報を処理する前段階のアルゴリズム

INPUT
magnitudeが0以上
Y
N
angleが0以上
Y
N
angleが0以上
Y
N
A = magnitude;
B = magnitude − angle;
B = magnitude;
A = magnitude + angle;
B = magnitude;
A = magnitude + angle;
A = magnitude;
B = magnitude − angle;
OUTPUT

ャネル間の余剰を削除するためにチャネルインタリービングという手法を使用します．

左/右の相関性が強い場合，polar mappingが優れているということです．

polar mappingを使う利点の一つは，拡散したイメージを作り出せることです．それは，与えられた周波数を使ってmagnitudeとangleの演算値で導出します．

magnitudeとangleどちらの情報も使用することによって，きめ細かい充実した情報を作り出すことが可能になります．

強い音は近い場所で発生する傾向があるので，大多数のステレオのイメージは極のmagnitude演算値だけで表現できます．

しかし，反響および拡散した音は，少ないmagnitudeで，かつ点在する光源のように複雑なangleの値によって音響心理学に支配される傾向があります．その場合，magnitude演算値だけでは正確な表現ができません．

しかしそこまでこだわっても，チャネル間でビットの漏出やクロストークが発生したら元も子もありません．

そこで，「多段式エンコーディング」を使用して，それを防止しています．また，三角波を丸めることでメリットを得ています．

三角波を丸めることで計算が複雑になることは，処理速度に悪影響を及ぼします．それがpolar mappingの問題点です．

補足すると，エンコーディングの過程で量子化を行っていることで，polar mapping情報と量子化情報は独立しています．

Vorbisは，polar mappingの利点を効果的に使用するため，加算/減算のみに依存します．そして，可能ならば左/右値の交換を行います．そして，1対1の写像によってangle/magnitudeを表すため，量子化する前に，また場合によっては量子化した後にも写像を使用します．

そのようにして，円ではなく平方ユニットにpolar mappingを行います．それがsquare polar mappingです．

magnitudeとangleから**図2**のように演算します．

● チャネルインタリービング

処理中に，再度polar mappingすることが可能です．そして，magnitudeとangleのベクトルを演算します．その結果，magnitudeのベクトル中のエネルギを集中し，angleのベクトルの中でコード化するべき情報の量を減らします．

residue backendで，これらのベクトルを独立してコード化することはビットレートに関係します．しかし，magnitudeとangleのベクトル間には，まだ暗黙の相関性があります．angleの振幅が，その対応するmagnitude値によって制限されます．

チャネルインタリービング処理によってベクトル量子化によってコード化される各ベクトルは，magnitudeとangleの価値が一致します．

● Vorbisが使用するステレオモデルについて

ステレオモデルには，以下のようなものがあります．

Dual Stereo

Lossless Stereo
Phase Stereo
Point stereo
Mixed Stereo

Dual Stereoは，二つのチャネルが完全に分かれている形式です．MP3は，この方式を利用しています．

Lossless Stereoは，Vorbisで使っている手法の一つです．前述のpolar mappingとチャネルインタリービングを使用しています．OggEnc 1.0以降では，高ビットレートのエンコードを行う場合に，このモードを使用します．

Phase Stereoはもっとも消極的な方式です．人間の耳が約4kHz以上の位相を判断できないという点が利用されるのです．実際そのとおりなのですが，人間の耳は機械ではないので理論どおりにはいきません．「Phase Stereo」は，angleのベクトルしか使用しません．この手法は，モノラルで録音されたSP盤を疑似ステレオにするために用いられてきたという歴史があります．

Point Stereoは，位相が違う信号を完全に除去します．この方式ではバランスのとれた反響になります．しかし，はっきりと音質の低下を感じます．とはいえ，はっきりとした位相を感じることが可能なので，ヘッドフォンを使って音楽を聴くことに向いています．

Mixed Stereoは，より高い周波数の領域で音質のよいモードを使用して，文字どおりこれまで挙げた方式を同時に使用します．よりロスのないカップリングを使用してコード化されるはずです．

現バージョンのVorbisは，Lossless Stereo，およびロスがないPoint Stereoから造られたMixed Stereoを使用します．

Phase Stereoはrc2エンコーダの中で使用されましたが，もはや単純すぎて現在は使用されていません．

なお，OggVorbisのライブラリを使ったプログラミングに関しては次回で触れます．

ライセンスの問題

さて，話題は変わります．音楽配信をする上で絶対に避けて通れない事柄の一つに「著作権」があります．

大手レコード会社と契約するミュージシャンが，自作自演をしてスタジオでそれを録音したとします．しかし，その音楽をミュージシャンが自由に配信したりすることはできません．なぜなら，「隣接著作権」というシステムがあるからです．それは録音技師にも，自分で打ち込みしなかった場合はMIDI作成者にも発生するのです．MP3配信の初期には，そうした問題からプロミュージシャンが自分の意志で音楽配信をすることは不可能でした．

1999年5月，P-MODEL（**図3**）がMP3配信でオーディエンスに音楽を届けるため，大手レコード会社との契約を破棄しました．そうして，「BitCash」などを使用してネット上で自らの作品を配信しました．それが国内におけるMP3配信の始まりでした．

それ以降，堰を切ったようにメジャー，マイナを問わず，MP3による音楽配信が一般化しました．

しかし，前回にも触れたとおり，ドイツのFraunhofer IIS-A社がMP3のライセンスを主張したため，音楽配信の現場では急速にMP3は廃れています．

そういう流れの中で，「オープンソース」の音楽版ともいえる「オープンオーディオライセンス」という概念が出現しました．

オープンソースの概念に関しては，今さら説明する必要はないかもしれませんが，詳細に関しては，

http://www.opensource.jp/index.html

で確認してください．

〔図3〕P-MODELのMP3配信ページ
(http://www.s-hirasawa.com/P-PLANT/)

〔図4〕Electronic Frontier Foundationのページ
(http://www.eff.org/IP/Open_licenses/eff_oal.html)

そして，オープンオーディオライセンスは，Electronic Frontier Foundation（**図4**）によって，2001年4月21日に文書で提言されました．

その概略は，配布される楽曲について，許可を得ることなくコピー・配布・利用・演奏してもよい代わりに，“必ず作者を明記すること”を要求するものです．自由にコピーをすることも許されます．配布できるということは，Napsterのようなシステムで合法的に音楽を広めることができるということです．

また，そこで手に入れた音楽をもとに新たなアレンジを行ったり，新たなメロディを付け加えることも無料で，かつ合法的にできるのです．

たしかに，オープンソースによってGNU/Linuxは現在のような優れたシステムになりました．ある人がバグを直し，ある人が機能を追加して……という具合に，時間ととも利便性が向上しました．

オープンオーディオライセンスの場合，リミックス，サンプリングなどの技法によって，新たな楽曲を作成したり，曲にボーカルトラックを入れるといったコラボレートができると面白くなるのですが……．

ただ，オープンソースを盗んで製品にしていたという事実や噂もあることなので，音楽に関しても同様のことが危惧されます．また，著作権が権利者の保護という，本来の目的から外れて「投資」の対象になったり，著作権自体が売買されている現状では，いろいろな問題が起こってくると思います．しかし，ネット上での音楽配信に関しての法的な問題点を解決する一つの道にはなると思います．

〔図5〕小西健司氏の音楽配信ページ（http://www.st.rim.or.jp/~ironbeat/music_wants-to_be-free/）

なお，DADAや4-D，P-MODELに参加していた小西健司氏がオープンオーディオライセンスに則して，Vorbisで楽曲の配布を行っています（**図5**）．

*　　　　　　　*

次回は，OggVorbisのライブラリを使ったプログラミングに関して，また音楽配信に関係する情報を紹介する予定です．

きし・てつお

CQ RISC評価キット
SH-4PCI with Linux 活用研究 1

GDB+DDDによる GUI対応デバッグ環境の構築

酒匂信尋

CQ RISC評価キット/SH-4PCI with Linuxでは，本ボードですぐに動作できるデバッガが用意されていない．そこで今回はターゲットボード上でgdbserverを動かし，PC/AT互換機のLinux上でGDB+DDDによるGUI対応のデバッグ環境を構築する．　（編集部）

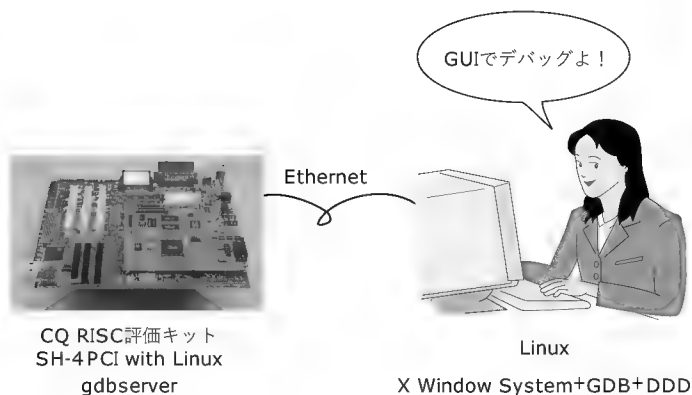
はじめに

組み込み機器のソフトウェア開発では，通常はICE（インサーキットエミュレータ）などのデバッガを使ってデバッグを進めます．最近のエミュレータはソースコードレベルでのデバッグ機能をもっているので，ITRON＋アプリケーションなど，実メモリ上で完結するものであれば，エミュレータだけを使ってデバッグ完了となります．

しかしLinuxを組み込む場合は，少し事情が違ってきます．LinuxはMMU（メモリマネジメントユニット）を使っているので，MMUの管理下でプログラムが動き出すと，一般的なエミュレータはMMUに対応していないので，この時点でお役御免となってしまいます．

パソコンでのLinuxは，高速なCPUと大容量のメモリとディスクのおかげで，セルフでコンパイル，デバッグができるわけですから，恵まれた開発環境をもっているといえます．

組み込みLinuxの場合はどうかというと，ターゲットのボードは限られた資源ですから，セルフでのコンパイル，ソースコードレベルのデバッグは望めません．頼りになるのはprintf文と，本来は別用途につけられたLEDなどです．一昔前であれば，これでも十分な開発環境でした．いまも，これで十分と思う人は多いようで，組み込みLinuxの開発環境の整備にはあまり熱心ではないようです．

〔図1〕システム構成

ここでは，限られた資源の組み込みボードでもソースコードレベルのデバッグができる環境を作ってみることにします．

1 システムの概要

CQ RISC評価キット/SH-4PCI with Linuxのボードには標準でEthernetを装備しています．そこで，Ethernet経由でリモートデバッグができる環境を前提とし，**図1**のような構成でいくことにします．ホストのLinuxパソコンにGDB＋DDD（Data Display Debugger，**図2**），そしてEthernet経由で接続されているターゲットでgdbserverを動作させます．

2 移植の手順

Linuxのディストリビューションの違い，クロス開発環境の違いによって，若干の違いがあると思いますので，今回の環境を

〔図2〕DDD（`http://www.gnu.org/software/ddd/ddd.html`）

〔図3〕ホスト側のGDBの作成

```
cd /opt/sh4src/gdb-5.2.1
mkdir buildc
cd buildc
../configure --build=i686-linux --host=i686-linux --target=sh4-linux --prefix=/opt/Embedix/tools
make
make install
```

〔図4〕ホスト側のGDBの起動メッセージ

```
[root@nobsun sh4src]# sh4-linux-gdb
GNU gdb 5.2.1
Copyright 2002 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "--host=i686-linux --target=sh4-linux".
(gdb) q
[root@nobsun sh4src]#
```

明確に示しておきます.

- ホストパソコン……ftp版TurboLinux7(フルインストール)
- クロス開発環境……CQ RISC評価キット/SH-4-PCIの付属品

● GDBの選択

最新のGDBは5.3です. x86のようなメジャーなプラットホームで使用するのであれば, 迷わず最新版を選択するのですが, SH-4用のGDBを作成するので, ある程度の不具合は想定されます. そこで, SH-4で実績のあるGDBをインターネットで探すとgdb-5.2.1のSH用パッチが, Linux on SuperH Projectのメンバーの小島さんのページにありました. gccのパッチなどで筆者もいつもお世話になっており実績があるので, 迷わずこのバージョンを移植することにします.

● DDDの選択

筆者のPC/AT互換機には, ftp版のTurboLinux7をフルインストールしたので, DDDもすでにインストールされています. 今回はそれを使用します. たぶん, どのディストリビューションでもコンパイル済みのDDDが収録されていると思いますが, もしない場合は, 次のサイトからダウンロードしてください.

- GDB-5.2.1
  http://ftp.gnu.org/gnu/gdb/
- SH用パッチ
  http://dodo.nurs.or.jp/~kkojima/index.html
- DDD
  http://ftp.gnu.org/gnu/ddd/

● GDBソースのセット

各自のクロス開発環境に合わせて都合のよいディレクトリにセットします. ここでは, /opt/sh4srcにダウンロードしたgdb-5.2.1.tgz, gdb-5.2.1-sh.diffを置いています.

```
# cd /opt/sh4src
# tar zxvf ./gdb-5.2.1.tgz
# cd  gdb-5.2.1
# patch -p1 < ../gdb-5.2.1-sh.diff
```

● クロス開発環境の設定

各自のクロス開発環境の設定を行います. 次の設定は, CQ RISC評価キット/SH-4PCIの場合の例です.

```
# export LD_LIBRARY_PATH=/opt/Embedix/
                            lwia1.0/usr/lib
# export PYTHONHOME=/opt/Embedix/
                            lwia1.0/usr/
# export PATH=/opt/Embedix/tools/bin:$PATH
```

● ホスト側のGDBの作成

まず, **図3**の手順で作成します. --prefix=/opt/Embedix/toolsは, でき上がったGDBのインストール先です. クロス開発環境に合わせたインストール先を指定します. sh4-linux-gdbを起動して, **図4**のような表示が出れば, 一応成功です.

● ターゲット側のGDBの作成

図5の手順で作成します. ホスト用GDBとの違いは--hostと--prefixの設定です. ../usr/localに作成されるので, 次のような手順でtarボールにします.

```
# cd ??/usr/local
# tar czvf - . > ../local.tgz
```

その後ftpなどでターゲットに転送し, 以下の手順で復元します.

```
# cd /usr/local
# tar xzvf ??/local.tgz
```

ターゲット上でgdbserverを起動して, **図6**のような表示が出れば一応成功です.

● 開発環境に依存したエラーの修正

通常は, ここまで説明した手順で作成できるはずですが, 今回の環境で, いくつかエラーがあったので, その対策です.

▶ホストパソコンの環境修正

クロス開発環境の再構築を行うと, **図7**のようなエラーで止まってしまいます.「mawkがないよ!」ということですので, 本来ならば, mawkをインストールするべきかもしれませんが, 機能的に互換性のあるgawkがインストールされていますので, こ

〔図5〕ターゲット側のGDBの作成

```
cd /opt/sh4src/gdb-5.2.1
mkdir build
cd build
../configure --build=i686-linux --host=sh4-linux --target=sh4-linux
             --prefix=/opt/lineo-BDK/KMC-BDK/images/image.sh4/root/usr/local
make
make install
```

〔図6〕ターゲット側のGDBの起動メッセージ

```
#
# gdbserver
Usage:  gdbserver tty prog [args ...]
        gdbserver tty --attach pid
Exiting
#
```

れを使用するように修正します.

```
# cd /bin
# ln -s gawk mawk
```

▶クロス開発環境の修正

gdbのconfigureで，**図8**のようなエラーとなってしまいます．クロス環境のldが参照するlibに`libncurses`がないようなので，作成します.

```
# cd /opt/lineo-BDK/KMC-BDK
# make nucurses
# cd /opt/lineo-BDK/KMC-BDK/build/
                                    packegs.sh4/lib
# cp -a libnc* /opt/Embedix/tools/
                                      sh4-linux/lib
```

▶GDBソースの修正

ターゲット側のGDB作成で，セルフのGDBは作成できましたが，今回の主目的であるgdbserverができませんでしたので，gdb-5.2.1に**図9**のようにパッチを当てます.

▶その他の注意点

configureを再度実行する場合は，`build`，`buildc`のディレクトリの下を削除してください.

3 簡単な使い方

● プログラムの作成

GDBでデバッグを行う場合，デバッグ情報が必要になります．これはコンパイル時に`-g`オプションをつけることで作成されます.

```
# sh4-linux-gcc -g -o p_test p_test.c
```

● プログラムの配置

プログラムはホスト側とターゲット側に必要ですからftpなどで転送し，`chmod`で実行権を与えます．実行権の設定のしかたは,

〔図7〕ホストパソコンの環境修正

```
make[2]: ここに入ります:  ディレクトリ `/opt/lineo-BDK/KMC-BDK/build/packages.sh4/sys/ncurses-5.2/man'
sh ./MKterminfo.sh ./terminfo.head ./../include/Caps ./terminfo.tail >terminfo.5
make[2]: ここから出ます:  ディレクトリ `/opt/lineo-BDK/KMC-BDK/build/packages.sh4/sys/ncurses-5.2/man'
cd include && make DESTDIR="/opt/lineo-BDK/KMC-BDK/build/packages.sh4/tmp/ncurses-5.2" all
make[2]: ここに入ります:  ディレクトリ `/opt/lineo-BDK/KMC-BDK/build/packages.sh4/sys/ncurses-5.2/include'
mawk -f MKterm.h.awk ./Caps > term.h
/bin/sh: mawk: command not found
make[2]: *** [term.h] エラー 127
make[2]: ここから出ます:  ディレクトリ `/opt/lineo-BDK/KMC-BDK/build/packages.sh4/sys/ncurses-5.2/include'
make[1]: *** [all] エラー 2
make[1]: ここから出ます:  ディレクトリ `/opt/lineo-BDK/KMC-BDK/build/packages.sh4/sys/ncurses-5.2'
Bad exit status from /opt/lineo-BDK/KMC-BDK/build/packages.sh4/tmp/rpm-tmp.53798 (%build)
make: *** [ncurses] エラー 1
[root@nobsun KMC-BDK]#
```

〔図8〕クロス開発環境の修正

```
checking compiler warning flags...  -Wimplicit -Wreturn-type -Wcomment -Wtrigraphs -Wformat -Wparentheses
-Wpointer-arith -Wuninitialized
checking for cygwin... no
checking for tgetent in -lncurses... (cached) no
checking for tgetent in -lHcurses... no
checking for tgetent in -ltermlib... no
checking for tgetent in -ltermcap... (cached) no
checking for tgetent in -lcurses... (cached) no
checking for tgetent in -lterminfo... no
configure: error: Could not find a term library
Configure in /opt/sh4src/gdb-5.2.1/build/gdb failed, exiting.
[root@nobsun build]#
```

〔図9〕GDBソースの修正

```
[root@nobsun gdb]# diff -urN configure.org configure
--- configure.org       Sat Dec  7 12:31:35 2002
+++ configure   Tue Dec 17 12:13:44 2002
@@ -8585,13 +8585,13 @@
 if test x"${target}" = x"${host}"; then
     echo $ac_n "checking whether gdbserver is supported on this host""... $ac_c" 1>&6
     echo "configure:8588: checking whether gdbserver is supported on this host" >&5
-    if test x"${build_gdbserver}" = xyes ; then
+#   if test x"${build_gdbserver}" = xyes ; then
        configdirs="${configdirs} gdbserver"
        SUBDIRS="${SUBDIRS} gdbserver"
        echo "$ac_t""yes" 1>&6
-    else
-       echo "$ac_t""no" 1>&6
-    fi
+#   else
+#      echo "$ac_t""no" 1>&6
+#   fi
 fi
```

```
# chmod +x ./p_test
```

とします．

● 操作方法

ターゲットのテストプログラムの転送が終わると，いよいよデバッグの開始となります．ターゲット側での操作でデバッグサーバを起動するには，

```
# gdbserver 192.168.1.10:2345 p_test
```

とします．192.168.1.10はホストのIPアドレス，2345は接続するポート番号，p_testがデバッグするプログラムです．正常に起動すると，

```
Process p_test created: pid = ???
```

ホスト側のGDBが起動されコネクションが確立すると，

```
Remote debugging from host 192.168.1.10
```

ホスト側から接続を切られると，

```
Killing inferior
```

と表示されます．操作の手違いなどで“Killing inferior”が表示された場合，再度gdbserverを起動します．

● ホスト側での操作

デバッガの起動は，次のようにします．

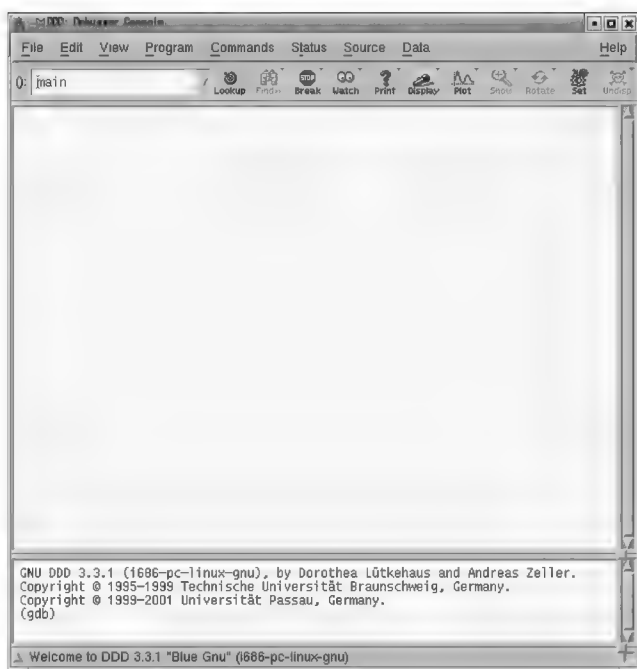
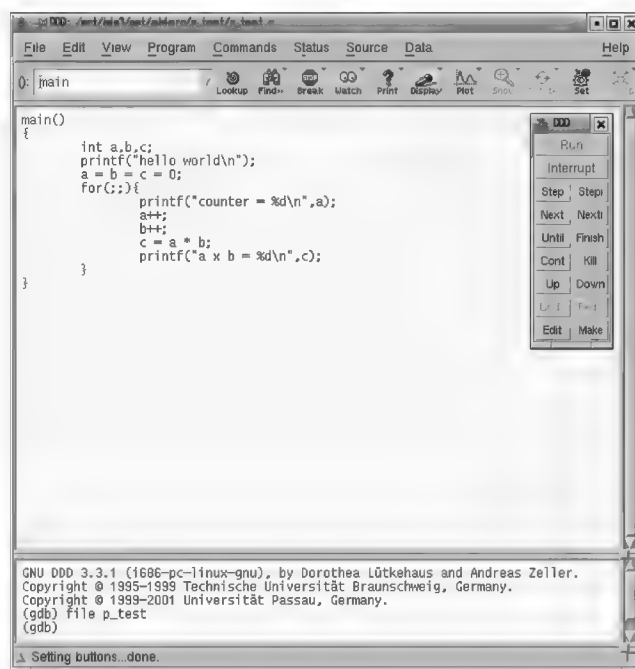
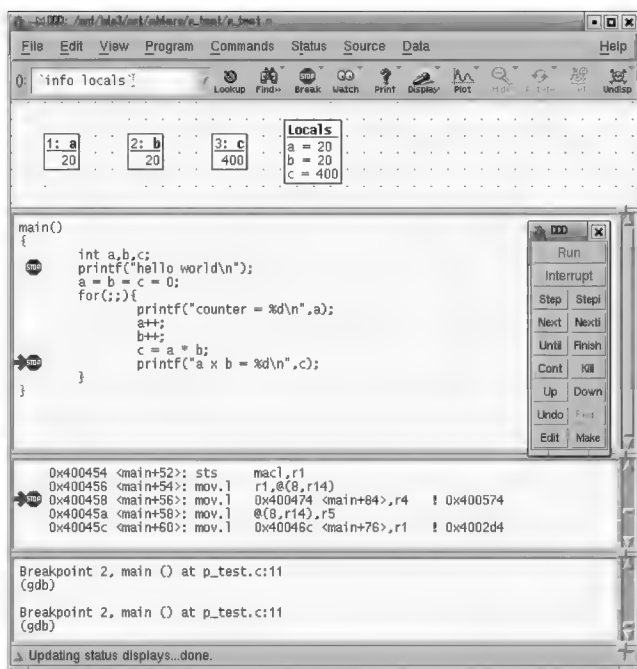
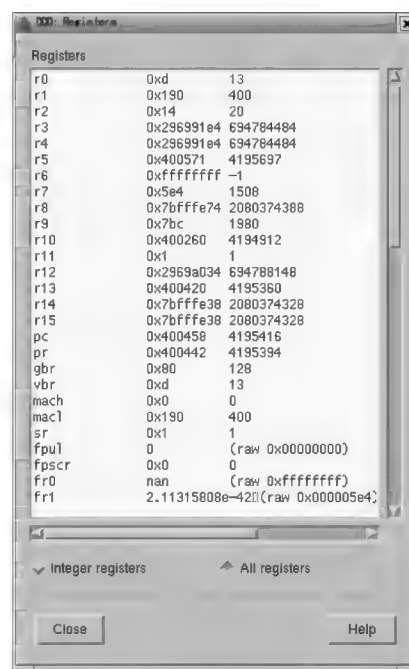
```
# ddd --debugger sh4-linux-gdb
```

正常に起動されると，**図10**のような画面が表示されます．シンボル情報の読み込みは，gdbコンソールより，

```
# file p_test
```

とします．読み込みが完了すると，**図11**の画面のようにソースコードが表示されます．

ターゲットへ接続するには，gdbコンソールより，

〔図10〕DDD起動画面

〔図11〕ソースコード読み込み時の画面

〔図12〕デバッグ時の画面

(a) 変数ウォッチのようす

(b) レジスタ表示のようす

```
# target remote 192.168.1.11:2345
```

とします．するとターゲット上に，

```
Remote debugging from host 192.168.1.10
```

と表示され，接続状態となります．

● デバッグの開始

DDDのWindow上からブレークポイントを設定し，gdbコンソールよりcコマンドでrunさせると，ブレークポイントで停止します．その後，1ステップずつ進めるのであれば，stepコマンド，次のブレークポイントまで走らせるのであれば，cコマンドをgdbコンソールより入力します．デバッグ中の画面を**図12**に示します．

● 操作上の注意点

GUIですから，操作方法を解説するより，適当にいじってもらうほうがよいと思います．GDBを単独で使う場合とgdbserverを通す場合で，若干のコマンドの違いがあります．DDDはGDB専用のようで，コマンドツールのボタンなどで使えないものがあります．使えないコマンドのボタンなどを押した場合，DDDがフリーズすることがあります．おとなしく使用するのであれば，現状でも問題はないと思いますが，本格的にリモートデバッグで使用する場合は，改善の余地があるようです．

さかわ・のぶひろ

ハッカーの常識的見聞録㉗

今月の常識 ノートPCがハイエンドデスクトップPCに近づいてきた！

■広畑由紀夫

☆「TOSHIBA DynaBook G6シリーズ」は，モバイルPentium4を搭載し，TVチューナやDVDドライブを搭載したハイエンドノートPCです．しかし，利点はそれだけではありません．今回は，DynaBook G6シリーズを購入したきっかけを報告します．

モバイルPentium4，2.2GHzというスペックのノートPC自体はDynaBook G6以前から発売されていますが，筆者はいままで使用していたDynaBook T5から，いきなり乗り換えてしまいました．いつもなら「少し早まったかな？」という感じがするのですが，今回はあまりそういう気がせず，むしろ「オプションを全部購入してしまおうか」とすら考えています．さて，ソフトウェア開発と個人的な趣味を一挙に満たす仕様とはどのようなものか，そして，今後に要求される仕様を考えてみます．

● DynaBook G6シリーズの基本的な仕様

ノートPCでは，多くの人が諦めていた「FINAL FANTASY XI」を「遊べる」動作速度で，しかもnVidia GForce4 460 Goによる高速描画かつ麗美な画質，そしてSPDIF出力可能なサウンド環境に，Bluetooth 1.1準拠，さらには802.11b無線LAN内蔵モデルという魅力的なスペックを備えています．USBも2.0対応で，IEEE1394も搭載しているため，オプションのブリッジメディアスロットを搭載した場合，その時点で内蔵スロットだけで読めないのはマイクロドライブと，SDメモリの約半分の大きさという xDピクチャーカードぐらいでしょう．

マルチスタイルベイのCF対応がType Iのため，マイクロドライブは使用できませんが，マイクロドライブやxDピクチャーカードは，PCMCIA/USB仕様のカードリーダを使うと割り切るのも一つの手段です．PCMCIAタイプなら，アダプタをスロットに挿入したままにしておくことも可能です．

●「内蔵HDDは一台」から解放された環境

仕様的に単体でもっとも優れたノートPCというだけなら，わざわざ話題にすることもありません．筆者が目をつけたのは，「マルチスタイルベイ」を使用して，複数の開発環境でソースコードを共有しつつ，平行開発できないかという点です．

Windows2000/XPになり，壊れやすいFAT32から，より安全なNTFSへ移行することが多くなると思いますが，NTFSに移行するとLinuxなどとのデュアルブートなどが難しくなります．さらに，FAT32がマスタブート領域（MBR）に割り当てられることも好ましくありません．

そこで，物理的にマルチスタイルベイにHDDか，512MバイトのCFメモリなどをFAT32で割り当て，こちらを複数OSで共有しやすいFAT32フォーマットにし，起動ドライブは，WindowsであればできるかぎりNTFSにします．さらに，Linuxなどと複数に割り当てる場合には，可能な限り互いに分離するか物理的に取り替え，ソースコードやデータ領域にマルチスタイルベイを使用することで，複数の開発環境で可能な限りデータやソースコードを1台で共有しようという算段です．

このように物理的に分離されていれば，システムドライブを取り替えることや，システムHDDはNTFSのままでKNOPPIX（CD-ROMでブートするLinux）をCD-ROMブートで使用しても安心です．この点においても，開発環境の幅が広がり，ネットワーク共有ドライブから解放された環境での開発やテストに利用できると思います．とくに，開発室から出て実地検証を行いながら作業を進める際，ネットワークが利用できなければ共有リソースにすらアクセスできないので，持ち歩きつつ複数の環境を共有できるという点は非常に役に立つと思います．もちろん，データドライブをネットワーク共有にすることもできます．

とくにノートPCでは，「内蔵HDDは1台」という一般的な仕様から，60GバイトのHDDを2台内蔵するという，デスクトップ環境に近い開発環境を持ち歩くことができるという点は重要だと思います．

● 筆者が求めていたもの

じつは，DynaBook T5を使用していていちばん気になっていたのはデザインでした．DynaBook G6を見ればわかるとおり（`http://dynabook.com/pc/catalog/dynabook/021015g6/index_j.htm`），非常に綺麗なクリアブルーの表面はとても魅力的です．しかし，それだけで選ぶはずもなく，ノートPCでHDDの増設が可能というオプションが，仕様的にとても魅力的でした．

筆者は，いつものごとくDynaBook G6 U22/PMEWを購入しに行ったのですが，11月発売にかかわらず品薄で在庫もなく，2002年度中は店頭の注文も停止状態でした．筆者の分は12月中旬にようやく店舗から入荷連絡があり，なんとか確保できました．なかなかに人気のようです．

皆さんも，マルチメディア用途以外に，開発用としても兼用できる便利なノートPCを使ってみませんか．

ひろはた・ゆきお　OpenLab.

CQ RISC評価キット

SH-4PCI with Linux 活用研究2

PCIデバイス対応デバイスドライバの作成法

竹内達也/田中 賢

CQ RISC評価キット/SH-4PCI with Linuxのボードには，PCI拡張スロットが用意されている．ここにPCI拡張ボードを実装してSH-4 Linuxから制御するためには，デバイスドライバが必要になる．ここではオンボードのLANコントローラと同じPCIデバイスを実装したPCI LANカードを使った場合と，ドライバが用意されていないテスト用PCIボードに対応した，デバイスドライバとテストプログラムの作成方法を解説する．（編集部）

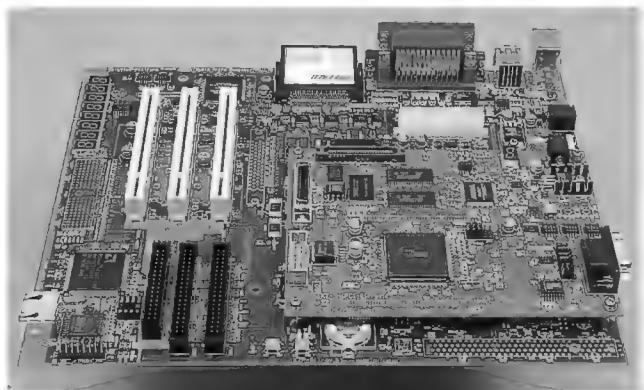
CQ RISC評価キット/SH-4PCI with Linuxの評価ボード（**写真1**）は，PCIバス上にPCに似た資源をもつマザーボードと，PCIホストコントローラを内蔵したSH-4（SH7751）を搭載したCPUボードの2枚1組からなるボードです．システムバスとしてPCIバスを採用したことで，PCI拡張スロットをもったPCライクなボードコンピュータとして活用することができます．

しかし，実際にPCI拡張スロットにPCIボードを実装し，SH-4用Linux上から制御するには，そのボードに対応したデバイスドライバが必要です．ここでは，CQ RISC評価キット/SH-4PCI with Linuxの評価ボードに対応したデバイスドライバの作成方法について解説します．

1 評価ボードのハードウェア

● PCIバスの構成

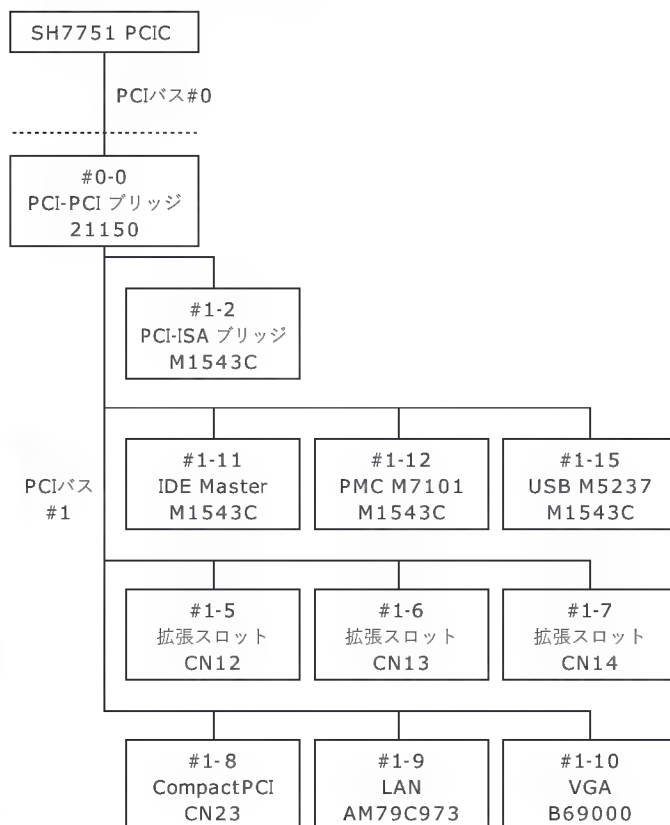
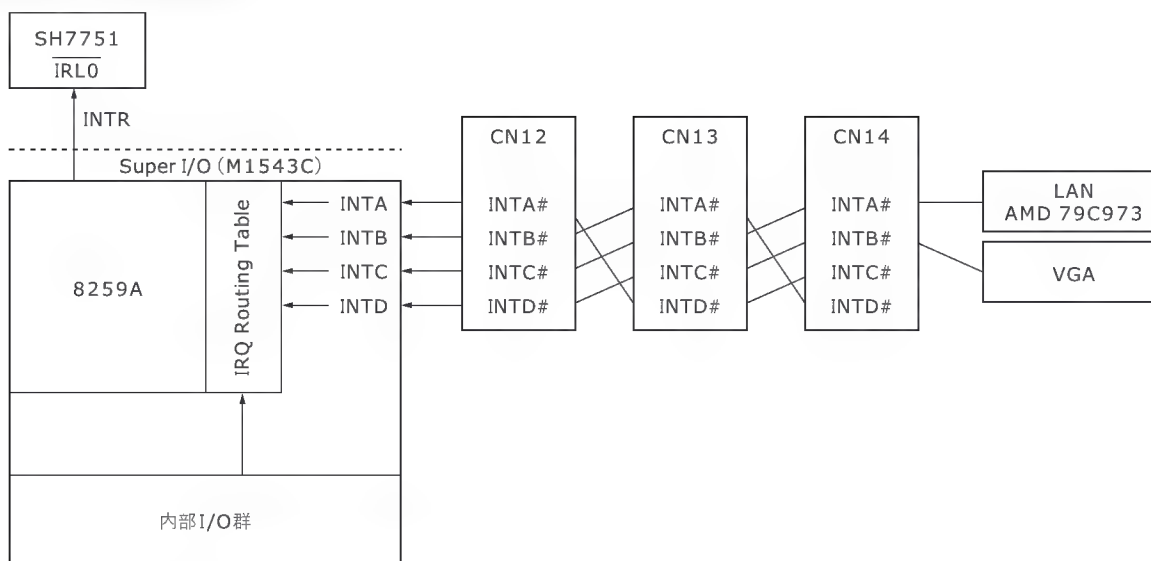
評価ボードの構成をPCIバスの観点から見ると，**図1**のようになります．まず，SH7751がホストデバイスとして存在し，バス#0に接続されています．このバスにはPCI-PCIブリッジとしてIntel21150が接続され，バス#1との橋渡しをしています．

バス#1にはオンボードのVGA，Ethernetコントローラや PCI-ISAブリッジのほか，PCI拡張スロットが三つ接続されています．PCI-ISAブリッジのM1543Cは，IDEやUSBのほかに割り込みコントローラ8259Aも内蔵しており，SuperI/Oの機能をあわせもっています．AM79C973（LAN）はオンボードのEthernetコントローラチップです．B69000はオンボードのビデオチップです．

〔写真1〕CQ RISC評価キット/SH-4PCI with Linuxの評価ボード

これらの資源がすべてPCIバス上に存在しています．これは，このマザーボードが特定のCPUに依存していないことを示します．そのため，マザーボード上に載せるCPUボードはPCIホストコントローラの機能さえもっていれば，機種やアーキテクチャは問いません．開発元である京都マイクロコンピュータでは，SH-4のほかにMIPSやARMなど，ほかのアーキテクチャのCPUボードも用意しています．

● 割り込み

評価ボードの割り込みは，**図2**のようになっています．注目すべき点は，PCI拡張スロットの割り込み信号（INTA#～INTD#）

〔図1〕評価ボードのPCIバスの構成

〔図2〕評価ボードの割り込みの構成

〔表1〕BIOSによるSH7751のPCICの初期化

レジスタ名称	アドレス	設定値	備考
PCICONF4	0xfe20_0010	0xfff0_0000	SH4内蔵PCICのベースアドレス
PCICONF5	0xfe20_0014	0xff00_0000	SH4のエリア3(SDRAM)のPCI側ベースアドレス
PCILSR0	0xfe20_0104	0x00f0_0000	SDRAMは16Mバイト空間
PCILAR0	0xfe20_010c	0x0c00_0000	SDRAMアドレスはSH4バスの0x0c00_0000に存在
PCICONF6	0xfe20_0018	0xfe00_0000	SH4のエリア4(SRAM)のPCI側ベースアドレス
PCILSR1	0xfe20_0108	0x00f0_0000	SRAMは16Mバイト空間
PCILAR1	0xfe20_0110	0x1000_0000	SRAMアドレスはSH4バスの0x1000_0000に存在
PCIBCR1	0xfe20_01e0	BSCのBCR1と同じ	
PCIBCR2	0xfe20_01e4	BSCのBCR2と同じ	
PCIWCR1	0xfe20_01e8	BSCのWCR1と同じ	
PCIWCR2	0xfe20_01ec	BSCのWCR2と同じ	
PCIWCR3	0xfe20_01f0	BSCのWCR3と同じ	
PCIMCR	0xfe20_01f4	BSCのMCRと同じ	
PCIMBR	0xfe20_01c4	0x0	
PCIIOBR	0xfe20_01c8	0x0	

(a) PCICへの設定値

PCIアドレス		アクセス先
0xfff0_0000～0xffff_ffff	I/Oアクセス	SH4内蔵PCICのローカルレジスタ
0xff00_0000～0xffff_ffff	メモリアクセス	SH4のエリア3(SDRAM) 0x0c00_0000～
0xfe00_0000～0xfeff_ffff	メモリアクセス	SH4のエリア4(SRAM) 0x1000_0000～

(b) PCIへのマッピングのようす

が，SuperI/O内の8259Aに接続されていることです．すなわち，すべての割り込みは1本の$\overline{IRL0}$信号になり，SH-4に接続されています．

このボード上で割り込みを使用するためには，SuperI/O内の割り込みルーティングテーブルと8259Aの正しい設定が不可欠です．

2 評価ボードのソフトウェア

評価ボードのSH-4 CPUボードはBIOSをもっており，起動時にボードがもっている各種資源を初期化します．PCIバス上の資源も初期化してくれるので，難しい設定をしなくてもほとんどの機能を使うことができます．

しかしながら，BIOSの動作を詳しく知ることが評価ボードに対応したソフトウェアを作成するには欠かすことができません．ここで，初期化時の動作を追ってみましょう．

● BIOSによるSH7751のPCIC初期化

評価ボードに電源が投入されると，BIOSはSH7751内蔵のPCIコントローラ(以下PCIC)を**表1(a)**のように初期化します．この初期化により，SH-4のローカル資源は**表1(b)**のようにPCI

〔表2〕BIOSによる21150の初期化

レジスタ名称	アドレスオフセット	設定値	備考
PrimaryBusNumber	+0x18	0x00	プライマリバス番号＝0
SecondaryBusNumber	+0x19	0x01	セカンダリバス番号＝1
SubordinateBusNumber	+0x1a	0x01	セカンダリバスにつながるバス番号の最大値
SecondaryLatencyTimer	+0x1b	0x40	レイテンシタイマ
I/OBaseAddressUpper16bits	+0x30	0x0000	
I/OBaseAddress	+0x1c	0x01	上位4ビット有効
I/OLimitAddressUpper16bits	+0x32	0x7fff	
I/OLimitAddress	+0x1d	0xf1	上位4ビット有効
MemoryBaseAddress	+0x20	0x0000	上位12ビット有効
MemoryLimitAddress	+0x22	0xfdf0	上位12ビット有効
PrefetchableMemoryBaseAddress	+0x24	0xfe01	上位12ビット有効
PrefetchableMemoryLimitAddress	+0x26	0xfe01	上位12ビット有効

(a) PCI-PCIブリッジの設定値

プライマリバス番号	0
セカンダリバス番号	1
PCIアドレス0～0x7fff_ffffに対するI/Oアクセス	バス#0からバス#1に転送
PCIアドレス0x0000_0000～0xfdff_ffffに対するメモリアクセス	バス#0からバス#1に転送
PCIアドレス0xfe00_0000～0xfe0f_ffffに対するメモリアクセス注	バス#0からバス#1に転送

(b) PCIへのマッピングのようす

注：このエリアはSH-4ローカルバスのSRAMにマッピングされているので，この設定は意味がない

〔表3〕BIOSによるオンボードPCI資源の初期化

アドレス	デバイス
0xbd00_0000～0xbd00_0fff	USB
0xbe00_0000～0xbe00_001f	AM79C973(LAN)
0xfd00_0000～0xfdff_ffff	B69000(LAN)

(a) メモリ空間

アドレス	デバイス
0xa000～0x101f	AM79C973(LAN)
0xa100～0xa13f	M7101 PMU
0xa180～0x1a9f	M7101 PMU
0xf000～0xf00f	M1543C IDE
0xffc8_0000～0xffc8_000f	RTC

(b) I/O空間

〔表4〕割り込みの初期化

IRQ番号	用途	IRQ番号	用途
IRQ0	TIMER	IRQ8	RTC
IRQ1	KeyBoard	IRQ9	UART2
IRQ2	8259Slave	IRQ10	あき
IRQ3	UART3	IRQ11	あき
IRQ4	UART1	IRQ12	Mouse
IRQ5	Parallel Port	IRQ13	あき
IRQ6	FDC	IRQ14	IDE Primary
IRQ7	INTC(LAN)	IRQ15	IDE Secondary

空間にマッピングされます．

SH-4はPCI空間へのアクセスを，メモリマップ上の特別な窓を通して行います．PCIメモリ空間に対しては0xfd00_0000～0xfdff_ffffの16Mバイトの窓が，I/O空間に対しては0xfe24_0000～0xfe27_ffffの256Kバイトの窓が用意されています．しかし，この窓の大きさはPCI空間より小さく，このままでは全領域をアクセスすることができません．これを解決するためにPCIMBRとPCIIOBRが用意されていて，PCI空間へ出力するアドレスの上位ビットを設定します．

当然ですが，このレジスタは，ユーザーが設定しなければなりません．PCIデバイスのベースアドレスレジスタを読み出して，上位バイトをこのレジスタに設定します．

● BIOSによる21150の初期化

BIOSはPCI-PCIブリッジである21150を**表2(a)**のように初期化します．これにより，プライマリPCIバス(SH-4側)とセカンダリPCIバスとは，**表2(b)**のようにブリッジされます．

● BIOSによるオンボードPCI資源の初期化

BIOSは，マザーボード上のPCI資源を**表3**のように初期化します．これ以外の資源(PCI拡張スロット)は，ユーザーが設定しなければなりません．ユーザーが資源の割り当てを行う場合には，これらBIOSが設定済みの資源と衝突しないように，資源を割り当てる配慮が必要です．

● BIOSによる割り込み資源の初期化

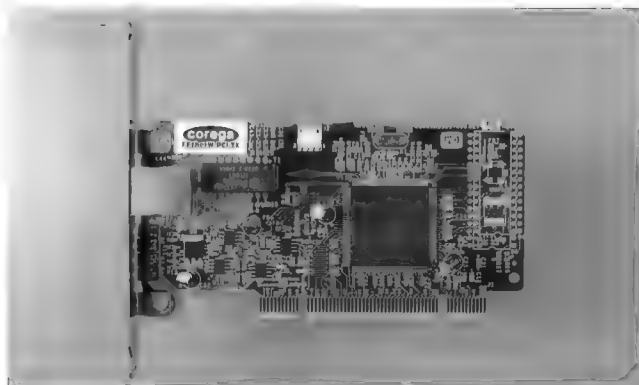
BIOSは，**表4**のように割り込みをマッピングします．これら，初期設定された割り込みは常に使われているわけではありませんが，PCI拡張スロットCN12，13で割り込みを使う場合は，空いているIRQ10，11，13を使うのが無難でしょう．CN14のINTA#はオンボードのLANのINTA#と共用になっているので，IRQ7を使うのがよいでしょう．

● BIOSがしてくれないこと

BIOSは，マザーボード上の拡張PCIスロット(CN12-CN14)に対してはいっさい関知していません．よって，拡張スロットを利用する場合には，ユーザーがすべての設定を行う必要があります．

ベースアドレスやコマンドレジスタを適切な値に設定し，また割り込みのルーティングやPCIボードへの割り込みラインレジスタの設定も必要になります．

ビデオカードを挿した場合には，オンボードVGAとの関係で，通常のPCIボードの初期化では行わないような初期化処理が増えます．PCIスロット上のビデオカードを使うには，まずオンボードのVGAを止めなければなりません．その後，PCIスロット上のビデオカードの設定を行い，動作を開始する必要があります．

〔写真2〕PCnet(AM79C973)搭載PCI LANボード
〔写真の製品はFEtherW PCI-TX(コレガ製)〕

● PCI拡張スロットを利用する場合の設定

マザーボード上のPCI拡張スロットを使うには，次のような処理が必要です．これらはBIOSでは行われないので，ユーザーが行う必要があります．

▶ベースアドレスの設定

PCIコンフィグレーション空間のベースアドレスレジスタに，ベースアドレスを設定します．PCIボードが要求しているすべての空間にベースアドレスを設定します．もちろん，他のPCI資源と重複しないような配慮が必要です．

▶基本クラス，サブクラスの確認

基本クラスとサブクラスは，ビデオカードを識別するのに重要です．マザーボードはオンボードでVGAをもっているため，不用意に拡張スロットのビデオカードを有効にすると，ビデオカードが複数存在することになり，正常に表示されないなどの問題が起こります．

▶割り込みの設定

SuperI/O内の割り込みルーティングテーブルを操作し，PCI拡張スロットの割り込み線を適切なIRQにルーティングされるように設定します．

その後，IRQ番号をPCIボードの割り込みラインレジスタに書き込んでおきます．デバイスドライバは，このレジスタを読むことにより，PCIボードにどのIRQ番号が割り当てられたかを知ることができるのです．

▶デバイスの有効化

コマンドレジスタの下位2ビットは，メモリ空間とI/O空間の有効化ビットです．どちらの空間を使うのかはカードによって異なりますが，これらのビットがセットされるとPCIボードは動作を開始します．

有効化ビットは，ユーザーがセットしなければなりません．

3 評価ボードでネットワークカードを使ってみる

ここまでの説明は，評価ボードの基本的な動作の説明だったので，以降はLinuxでPCI拡張スロットを使ってみます．

〔表5〕ネットワークカードに割り当てるリソース

PCI拡張スロット	CN13
メモリ空間	0xbf00_0000～0xbf00_001f
I/O空間	0xb000～0xb01f
割り込み	IRQ10

〔リスト1〕ネットワークカードのリソース割り当て処理

```
pci cfgwr(0x01,0x06,PCI BASE ADDRESS 0,0x0000b000);
pci_cfgwr(0x01,0x06,PCI BASE ADDRESS 1,0xb0000000);
pci cfgwr(0x01,0x02,0x48,pci cfgrd(0x01,0x02,0x48) | 0x00000030);
outb(inb(0x4d1) | 0x04, 0x4d1);
pci cfgwr(0x01,0x06,PCI INTERRUPT LINE,0x0a);
```

まずは市販のネットワークカードを例に，PCIボードを動作させてみます．ここではオンボードのネットワークコントローラと同じAMDのPCnetを搭載したPCIボードを用意します(**写真2**)．

この場合，同じチップを使ったオンボードのネットワークはすでに動いているわけですから，デバイスドライバを新規に開発する必要はありません．デバイスドライバは動作実績があるのですから，それをそのまま流用し，後はメモリやI/O，割り込みの割り当てを正しく行えば動作します．

● 注意点

PCIボードの資源は，オンボード資源が使っていない空き領域におかなければなりません．今回は，使うスロットを固定して，カーネルソースコードに直接マッピング情報を記述する方法をとります．三つあるPCI拡張スロットのどのスロットに差し込んでも動作するようにするには，バスをスキャンして，未割り当ての資源を自動設定する方法が必要ですが，ここでは解説しません．

● 設定変更点

表5のような設定で動作させることにしましょう．

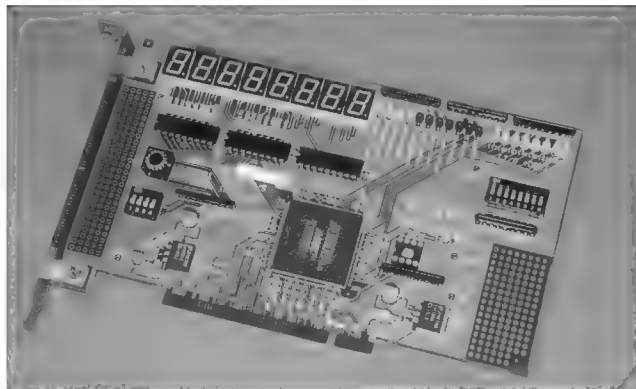
まずI/O空間ですが，AM79C973の場合はPCIコンフィグレーション空間のベースアドレス0に設定します．続いて，メモリ空間をベースアドレス1に設定します．

PCI拡張スロット(CN13)のINTA#はSuperI/OのINTB#に接続されています．INTB#をIRQ10にルーティングします．また，PCIの割り込みはレベル割り込みなので，IRQ10をレベル割り込みに設定します．

最後に，PCIボードの割り込みラインレジスタに割り込み番号10を書き込みます．

これらを`arch/sh/kernel/setup_kzp01.c`の`setup_kzp()`内に記述します(**リスト1**)．

`arch/sh/kernel/setup_kzp01.c`の`setup_kzp()`のAM79C973の設定では，BCR25とBCR26にも設定を行っていますが，PCIボード上に実装されたAM79C973では，この処理は必要ありません．これらのレジスタはMACアドレスなどの情報と一緒に，PCIボード上のROMから読み込まれて自動設定さ

〔写真3〕TE001の外観

れています．評価ボードでは，このAM79C973用の初期化用のROMをもたないので，ソフトウェアで設定しているのです．

● 動かしてみる

いつものように，カーネルを構築して起動してみます．コンソールが立ち上がったところでログインし，次のようなコマンドを入力します．

```
# ip addr 192.168.1.100/24 dev eth1 brd
                    192.168.1.255 scope global
# ip link set 192.168.1.100 up
# ip route append default via 192.168.1.254
                                    metric 30001
```

この設定例は，自分のIPアドレスが192.168.1.100/24，ゲートウェイが192.168.1.254の場合なので，ユーザーの環境に合わせて値を変えてください．

以上の操作は，/etc/rc.d/init.d/S20networkを修正して実行しても同じです．そして，

```
# ping 192.168.1.1
```

などとすれば動作を確認できます．

4 新規ドライバの作成法

● 今回のターゲットボード

次に，汎用PCIボードを使ってデバイスドライバとテストプログラムを作ってみます．

今回は，TE001(田中製作所製)をターゲットとして取り上げてみます．このボードにはPCIデバイスドライバの動作を確認するための機能が盛り込まれており，デバイスドライバ作成の手助けになります．具体的には，ベースアドレスレジスタやコマンドレジスタが設定されているかどうかのLED表示，I/O空間のLED表示とスイッチ入力，メモリ空間のLED表示，定期的な割り込み発生とマスクなどの機能をもち，測定器がなくともPCIボードの動作をある程度目視することが可能です．

TE001のボードの外観を**写真3**に，仕様を**表6**に示します．このボードのメモリ空間の先頭8バイトには，7セグメントLEDに表示するレジスタが割り当てられているので，定時割り込みと併用して，テストプログラムを作ってみましょう．

〔表6〕TE001の仕様

メモリ空間	ベースアドレス0に256バイト
I/O空間	ベースアドレス1に4バイト
割り込み	1秒ごとの定期割り込み→INTA#

〔表7〕TE001のリソース割り当て

メモリ空間	0xbe10_0000～0xbe10_00ff
I/O空間	0xa800～0xa803
割り込み番号	IRQ10
スロット	CN13(PCIデバイス番号0x06)

● デバイスドライバの方針

メモリ空間へのリードとライトのためにte001_read()とte001_write()を実装します．te001_write()は，割り込みが有効に設定されているときは，動作を割り込みに同期させるようにしましょう．te001_write()の入り口で，interruptible_sleep_on()を実行し，次の定時割り込みまでブロックします．

I/O空間へのリードとライトは，ioctl()経由で行うようにします．こちらは，割り込みに関係なく常にリードライト可能とします．

定時割り込みは，割り込みハンドラを用意し，割り込み発生でwake_up_interruptible()を実行し，ブロックしているwrite()を呼び起こします．

これらの組み合わせで，割り込み無効のときには自由にメモリ空間をアクセス可能とし，割り込み有効時には定時割り込みのタイミングでメモリに書き込む，というデバイスドライバを作成することにします．

● カーネルソースの修正

まず，TE001のための設定をしなければなりません．**表6**によるPCIボードの仕様を参照して，PCI資源の設定の空き領域をTE001に割り当てます(**表7**)．

先ほどと同様に，PCI拡張スロットのCN13に挿すことにすると，CN13のINTA#はSuperI/OのINTB#に接続されているので，INTB#をIRQ10にルーティングし，IRQ10をレベル割り込みに設定します．

TE001にリソースを割り当てるには，arch/sh/kernel/setup_kzp01.cのsetup_kzp()を**リスト2**のように記述します．カーネル再構築の際には，ローダブルモジュールを有効にしておきます．

今回，これらの初期化は，スロットを固定しアドレスも手動で設定していますが，kernel内の機能を使って自動割り当てすることも可能です．ベースアドレス未割り当てのPCIボードは自動的にアドレスを割り振ってくれるので，それほど難しい話ではありません．ただしその際には，PCIボードのコマンドレジスタのメモリやI/O空間イネーブルビットを初期化するのを忘れないでください．

〔リスト2〕TE001のリソース割り当て処理

```
pci cfgwr(1, 0x06, 0, PCI BASE ADDRESS 0, 0xbe100000);
pci cfgwr(1, 0x06, 0, PCI BASE ADDRESS 1, 0x0000a800);
pci cfgwr(1, 0x06, 0, PCI COMMAND, 0x02800003);
pci cfgwr(1, 0x06, 0, PCI INTERRUPT LINE, 0x0a);
pci cfgwr(1, 0x02, 0, 0x48, pci cfgrd(1, 0x02, 0, 0x48) | 0x00000030);
outb(inb(0x4d1) | 0x04, 0x4d1);
```

実際のデバイスドライバのソースは，**リスト3**のようになります．

● テストプログラム

テストプログラムは，対話形式で動作するものとし，**表8**のようなコマンドを実装します．実際のプログラムは，**リスト4**(p.133)のようになります．

まずは，デバイスドライバをロードします．

```
# insmod /lib/modules/2.4.5/char/te001.o
```

正しくロードできたかどうか，dmesgで確認してみます．

```
# dmesg
```

これで“te001 : module loaded”の文字列を確認できれば，第一段階は通過です．確認できなかった場合には，デバイスドライバが正しく作られているか，PCIボードは正しく挿されているかを確認してみてください．

続いて，テストプログラムを起動してみます．

```
# /usr/local/bin/test
```

正常であればプロンプト‘>’を表示し，コマンド待ち状態になります．起動に失敗した場合は，dmesgでメッセージを確認してみてください．

次に簡単な使い方を説明します．なお，指定する番地は，TE001に割り当てられたメモリやI/Oアドレスの先頭からのオフセットです．

```
>r 0 10
```

これで，メモリの内容を0x00番地から0x10バイト読み出し表示します．

```
>w 1 55
```

これでメモリの0x01番地に0x55を書き込みます．

```
>i 0
```

これでI/O空間の0x00番地を読み出して表示します．

```
>o 0 1
```

これでI/O空間の0x00番地に0x01を書き込みます．

```
>a
```

これで自動書き込み動作を開始します．自動書き込みモードは10回書き込んだところで終了します．このとき，TE001の設定をメモリ領域のビットパターン表示にしておくと，LEDに表示されるパターンが1秒ごと更新されていくようすが見られます．

〔表8〕テストプログラムのコマンド

コマンド	機　能
r [adr] [count]	メモリ領域リード
w adr data	メモリ領域バイトライト
i [adr] [count]	I/O領域リード
o adr data	I/O領域ライト
a	割り込み有効にし，メモリ領域に連続書き込み

終了する場合は，

```
>ctrl-d
```

とします．

おわりに

以上，駆け足ではありましたが，CQ RISC評価キット/SH-4PCI with Linuxの評価ボードのPCI拡張スロットを活用する方法について説明しました．評価ボードとLinux，そしてPCIの基本的な知識があれば，わずかな手間でPCI拡張スロットを使うことができるということをおわかりいただけたと思います．

本格的な応用は皆さんにおまかせするとして，この記事がこの評価ボードでPCI拡張ボードを使いたいのに躊躇されている方の手助けとなればうれしい限りです．

参考文献

1)『SolutionPlatform SH-4PCI(SH7751)取扱説明書』，京都マイクロコンピュータ(株)
2) PCI Local Bus Specification Revision2.2
3) TECH I Vol.3，『PCIデバイス設計入門』，CQ出版(株)
4) *LINUX DEVICE DRIVERS*, Allesandro Rubini & Jonathan Corbet (O REYLLY).

たけうち・たつや　(株)リネオ
たなか・けん　(有)田中製作所

〔リスト3〕デバイスドライバのソース

```
#define MEM SIZE  0x100
#define IO SIZE   0x04

#define TE001 MAGIC    0xc0
#define TE001 IORD IOR(TE001 MAGIC, 0, arg)
#define TE001 IOWR IOW(TE001 MAGIC, 1, arg)
#define TE001 INT ENABLE     IOW(TE001 MAGIC, 2, arg)
#define TE001 INT DISABLE    IOW(TE001 MAGIC, 3, arg)
```

(a) TE001.H

```
/*
 * te001.c
*/

#ifndef   KERNEL
  #define   KERNEL
#endif

#ifndef MODULE
  #define MODULE
#endif
```

(b) TE001.C

〔リスト3〕デバイスドライバのソース（つづき）

```
#define   NO VERSION

#define EXTERN INLINE extern   inline

#include <linux/module.h>
#include <linux/version.h>

#include <asm/io.h>
#include <asm/uaccess.h>
#include <asm/segment.h>

char kernel version[] = UTS RELEASE;

#include <linux/sched.h>
#include <linux/proc fs.h>
#include <linux/pci.h>
#include <linux/fs.h>
#include <linux/vmalloc.h>
#include <linux/compatmac.h>
#include "te001.h"

unsigned char bus, func;
struct pci dev *dev;
unsigned short vendor = 0x1172, id = 0xe001;
unsigned int TE001 MAJOR = 0;

unsigned long mem start;
unsigned long io start;
int int count;
char irq_no;
void *dev id;
DECLARE WAIT QUEUE HEAD(te001 q);
int int enable;

/* 割り込みハンドラ */
void te001 interrupt(int irq, void *dev id, struct pt regs *regs)
{
  if (!(inb(io start + 2) & 0x01))
/* TE001が割り込みを発生していなければ，何もしない */
    return;

  /* 割り込み要因クリア */
  outb(inb(io start + 2) & ~0x01, io start + 2);

  writeb(int count, mem start + 0x08);

  wake up interruptible(&te001 q);

  int count++;
}

/*  ドライバオープン処理 */
int te001 open(struct inode *inode, struct file *file)
{
  unsigned int tmp;

  if (!pci present()) {
    printk(KERN DEBUG "te001 : No PCI bios present\n");
    return -ENODEV;
  }
  if ((dev = pci find device(vendor, id, dev)) == 0) {
    printk(KERN DEBUG "te001 : No Device found. vendor = 0x%x, id
                                          = 0x%x\n", vendor, id);
    return -ENODEV;
  }

  pci read config dword(dev, PCI BASE ADDRESS 0, &tmp);
  mem start = tmp & 0xfffffffc;

  pci read config dword(dev, PCI BASE ADDRESS 1, &tmp);
  io start = tmp & 0xfffffffc;

  /* biosが設定した割り込み番号を調べる */
  pci read config byte(dev, PCI INTERRUPT LINE, &irq no);

  if (request irq(irq_no, te001 interrupt, SA INTERRUPT,
        "te001", NULL) != 0) {
    /* Error */
    printk(KERN DEBUG "te001 : request irq() error\n");
    return -EIO;
  } else {
    printk(KERN DEBUG "te001 : request irq() succeeded\n");
  }

  printk(KERN DEBUG "te001 : device found. vendor = 0x%x, id
                                          = 0x%x\n", vendor, id);
  printk(KERN DEBUG "te001 : mem = 0x%08x, io = 0x%04x, irq_no
                                                     = 0x%02x\n",
         (int)mem start, (int)io start, irq no);
  MOD INC USE COUNT;
  return (0);
}

/*  クローズ処理 */
int te001 release(struct inode *inode, struct file *file)
{
  printk(KERN DEBUG "te001 : release\n");
  free irq(irq_no, dev id);
  MOD DEC USE COUNT;
  return (0);
}

/*  メモリ空間リード */
ssize t te001 read(struct file *file, char *buf, size t len,
                                                   loff t *f pos)
{
  unsigned long adr;
  unsigned long val;
  size t i;

  if (len > MEM SIZE)
    len = MEM SIZE;
  if (*f pos + len > MEM SIZE)
    len = MEM SIZE - *f pos;

  for (i = 0; i < len; i++) {
    val = readb(mem start + *f pos + i);
    copy to user(buf++, &val, 1);
  }

  *f pos += len;
  return (len);
}

/* メモリ空間ライト */
ssize t te001 write(struct file *file, const char *buf,
                                     size t len, loff t *f pos)
{
  char val;
  size t i;

  if (len > MEM SIZE)
    len = MEM SIZE;
  if (*f pos + len > MEM SIZE)
    len = MEM SIZE - *f pos;

  if (int enable)
    /* 割り込みが有効になっていれば，次の割り込みまでブロックする */
    interruptible sleep on(&te001 q);
  for (i = 0; i < len; i++) {
    copy from user(&val, buf++, 1);
    writeb(val, mem start + *f pos + i);
  }

  *f pos += len;
  return (len);
}

/* ioctl
 *  I/O空間のアクセスと，割り込み発生許可・停止に使用
 */
int te001 ioctl(struct inode *inode, struct file *file,
         unsigned int cmd, unsigned long arg)
{
  char adr, data;
  int tmp;

  switch (cmd) {
  case TE001 IORD:
    get user(adr, (char*)arg);
    put user(inb(io start + adr), (char*)arg);
    break;
```

（b）TE001.C（つづき）

〔リスト3〕デバイスドライバのソース（つづき）

```
  case TE001 IOWR:
    get user(tmp, (int*)arg);
    data = tmp & 0xff;
    adr = (tmp >> 8) & 0xff;
    outb(data, io start + adr);
    break;
  case TE001 INT ENABLE:
    outb(inb(io start + 2) | 0x02, io start + 2);
    int enable = 1;
    break;
  case TE001 INT DISABLE:
    outb(inb(io start + 2) & ~0x02, io start + 2);
    int enable = 0;
    break;
  default:
    return -EINVAL;
  }
  return 0;
}


struct file operations te001 fops = {
  ioctl: te001 ioctl,
  read:  te001 read,
  write: te001 write,
  open:  te001 open,
  release:   te001 release,
};

/*  モジュールの初期化 */
int init module(void)
{
  int result;

  result = register chrdev(TE001 MAJOR, "te001", &te001 fops);
  if (result < 0) {
    printk(KERN DEBUG "te001.o : unable to get major number.\n");
    return 0;
  }
  TE001 MAJOR = result;
  printk(KERN DEBUG "te001 : module loaded\n");
  return 0;
}

void cleanup module(void)
{
  unregister chrdev(TE001 MAJOR, "te001");
  return;
}
```

（b）TE001.C（つづき）

〔リスト4〕テストプログラム（PCI.C）

```
/*
 * pci.c
 *
*/

#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>
#include <linux/fs.h>
#include "te001.h"

int arg;

/* メモリ領域に書き込むビットパターン */
char pattern[] = {
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x08, 0x1c, 0x7c, 0x7e, 0x7f, 0x7e, 0x7c, 0x1c,
  0x08, 00
};

void do command(int fd, char *cmd, unsigned int adr,
                                               unsigned int data)
{
  int offset = 0, count;
  unsigned char buf[MEM SIZE];
  int i, len, arg;

  if ((cmd[0] == 'i') || (cmd[0] == 'r'))
    if ((count = data) == 0)
         count = 1;;

if (cmd[0] == 'r') {
    /* メモリ領域リード */
    if (adr >= MEM SIZE)
         adr = MEM SIZE - 1;
    lseek(fd, adr, 0);

    if (count + adr > MEM SIZE)
         count = MEM SIZE - adr;

    read(fd, buf, count);
    for (i = 0; i < count; i++) {
         if ((i % 16) == 0)
              printf("\n%08x : ", adr + i);
         printf(" %02x", buf[i]);
    }
} else if (cmd[0] == 'i') {
    /* I/O領域リード */
    printf("io : 0x%02x =", adr);
    ioctl(fd, TE001 IORD, &adr);
    printf(" 0x%02x\n", adr);
} else if (cmd[0] == 'w') {
    /*  メモリ領域ライト */
    lseek(fd, adr, 0);
    write(fd, &data, 1);
} else if (cmd[0] == 'o') {
    /* I/O領域ライト */
    arg = (adr << 8) | data & 0xff;
    ioctl(fd, TE001 IOWR, &arg);
} else if (cmd[0] == 'a') {
    /* 割り込みを有効にし，連続書き込み */
    ioctl(fd, TE001 INT ENABLE);
    for (i = 0; i < 10; i++) {
         lseek(fd, 0, 0);
         write(fd, &pattern[i], 8);
    }
    ioctl(fd, TE001 INT DISABLE);
  }
  printf("\n");
}

int main(int argc, char **argv)
{
  FILE *input stream = NULL;
  char cmd[10], buf[BUFSIZ];
  unsigned int adr, data;
  int fd;

  if ((fd = open("/dev/te001", O RDWR)) == -1) {
    fprintf(stderr, "Can't open /dev/te001\n");
    exit(2);
  }

  while (printf("> "), fgets(buf, sizeof buf, stdin) != NULL) {
    cmd[0] = adr = data = 0;
    sscanf(buf, "%s %x %x %x", cmd, &adr, &data);
    do command(fd, cmd, adr, data);
  }

  close(fd);
}
```

論理ドライブの読み取りや
プラグイン情報の調査方法を紹介

外部メディアのバックアップ プログラムを作成する

広畑由起夫

本稿では，先月号（2003年2月号）の「ハッカーの常識的見聞録」で紹介した，ディジタルカメラ用メモリのバックアップをPCで行うプログラムの解説を行います．前号で紹介したように，メモリカードのフォーマットと再利用で困った状況に陥りました．実際，Windows XP上でFAT32フォーマットされたメディアの再フォーマットができない場合，通常なら新品のメディアを買ってきて，それまで使用していたメディアはパソコン専用にまわってしまうでしょう．本稿で紹介するのは，そのようなことを少しでも防ぐためのプログラムです．

また，プログラムの解説を通して，論理ドライブの読み取りやデバイスのプラグイン情報の調査方法などについて説明します．

● ソフトウェアの機能

このソフトウェアは，CFメモリカードなど，比較的小容量なFAT/FAT32ドライブに対して，論理フォーマット単位でのバックアップと修復，およびバックアップ時のファイルのコピー，もしくは移動を行います．論理フォーマットの修復はイメージモードのみで行うことができ，バックアップからのファイル単位での修復はできません．

● バックアップ可能なメディア

Windoows XPでFAT/FAT32など，FAT系として認識されるドライブおよびメディアを対象としています．大容量メディアに対しては，バックアップ時間や容量の都合上，バックアップや修復は行わないようにしています．FAT/FAT32で扱える容量を越えるメディアにアクセスする場合には，ソースコードの中でFAT/FAT32を検出している部分を削除して再構築することにより，一部のデバイス（論理フォーマット情報が取得可能なもの）に対してはバックアップ可能になりますが，バックアップ時間などを考えるとすすめられません．また，デバイスI/Oで，論理アクセスが禁止されている場合にはアクセスできないので注意してください．

イメージバックアップでは，メディアの論理フォーマット情報（FAT情報）全体とファイルの内容をバックアップします．同時に，ファイルモードでのバックアップを指定しておくことで，バックアップ対象のドライブを階層ごとにバックアップします．標準設定では，イメージとファイル両方をバックアップすることで，メディアの情報とファイル単位での情報の保存を行います．

万が一，メディアを物理フォーマットしてしまった場合などには，修復モードを選択することで，イメージバックアップした際に作成されるPBIファイルから元のメディアに論理フォーマット情報を戻すことができます．

● 使用上の注意

本稿で紹介するプログラムは，たしかにバックアッププログラムですが，何らかの原因でデータ破損やロストなどが起こったとしても保証はしかねます．重要なデータは，複数の方法でバックアップをとったうえ，バックアップができているかどうかの確認をしっかりと行ってください．

また，修復モードで元のメディアに論理フォーマット情報を書き戻すことができますが，その際に同容量でないメディアであれば警告が出るようになっています．メディア情報が異なっても論理フォーマットを書いて何とかトラブルを解消したいという特殊な場合以外には使用しないでください．容量の異なるメディアへの書き込みは，メディアの破損などにつながるおそれがあります．

使用方法

1）事前の準備

① 使用したいドライブやメディアが，使用するPCの環境でドライブとして認識され，プラグインした際に正常に使用できるかどうかを確認しておいてください．また，プラグインしたときのドライブ情報はメモなどに控えておいてください．

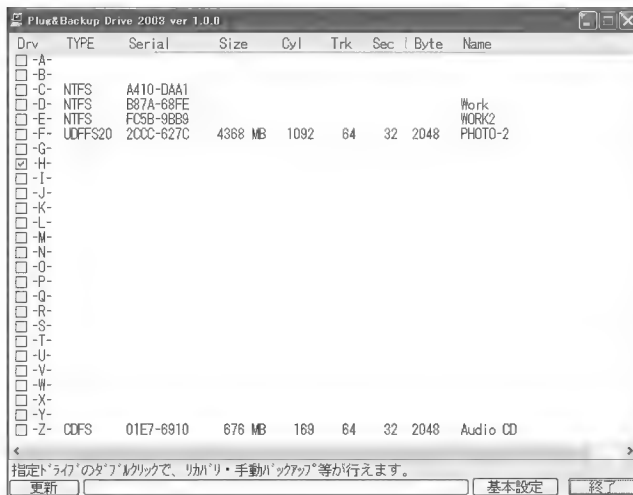
② PDB03（Plug&Backup Driver 2003）を起動し（**図1**），①で認識したメディアのドライブレターにチェックをしてください．チェックをすることで，プラグインされた際の自動バックアップ検査対象ドライブとなります．このとき，①のメディアが挿入されている場合には，容量やドライブのフォーマット情報が表示されます．フォーマットがFAT/FAT32以外の場合には，バックアップおよびリカバリ対象にはなりません．

2）バックアップ

ドライブにメディアが差し込まれたり，ドライブがプラグインされて自動認識されたとき（**図2**）に，それがバックアップ可能なメディアであれば，バックアップを開始します（**図3**）．

〔図1〕起動画面

〔図2〕プラグインしたときの画面

〔図3〕プラグインされ，バックアップ中の画面

バックアップする先は，標準設定でPDB03の起動フォルダの下に「BACKUP」フォルダが作成され，その中にメディアのシリアル番号，そして日時単位のバックアップファイルやファイル単位でのバックアップ内容が記録されます．バックアップの基本となるフォルダの変更は，5)の項を参照してください．

●バックアップフォルダの構造

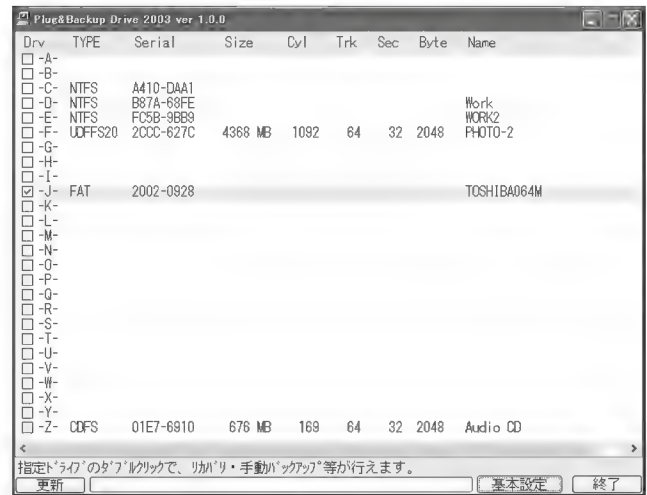
まず，ルートの下に，ボリューム名のフォルダが作成され，次に，ボリュームシリアルのフォルダが作成されます．この領域に，拡張子「PBI」でイメージバックアップファイルが作成日時のファイル名で保存されます．さらに，その下にドライブ名がフォルダとして作成され，そこにファイル単位のバックアップが保存されます．

このように，ボリューム名を種別/ボリュームシリアルでメディア番号，日時別のバックアップを自動的に行うようになっています．ファイルのバックアップを同時に行う設定であれば，ファイルの内容からどのバックアップファイルを使用すべきかなどの判断がつくことでしょう．

バックアップのルートを変更した場合，そのすぐ下にボリューム名のフォルダが作成されるので，この点に注意してください．また，ボリュームラベルが設定されていない場合は，‘[No Name]’フォルダが割り当てられ，他のボリュームと区別できるようにしています．

3) リカバリ(修復)

① イメージバックアップがなされている場合には，バックアップを行いたいドライブをダブルクリックし，「Recovery」を選択します．そして，バックアップ保存フォルダの中にあるイメージファイル(拡張子PBI)を指定して，修復を開始するだけです．

② 容量の異なるメディアへの書き込みは，正常にリカバリできない場合や，メディアの破損などにつながる場合があるので，行わないようにしてください．

4) 指定ドライブバックアップ

① プラグインによる自動バックアップではなく，すでにドライブが存在するメディアからの手動バックアップは，バックアップするドライブ(FAT/FAT32限定)をダブルクリックし，イメージバックアップ，もしくはファイルバックアップを選択することで手動でのバックアップが可能です．

自動バックアップした後，メディアを交換せずにファイル操作などを行った結果をバックアップしたいときなどに利用してください．

② ボリュームシリアル番号の変更

ボリュームシリアル番号はランダムに割り当てられる32ビットの数ですが，メディア情報を直接書き換えることでボリュームシリアル番号を編集できる機能を付けることにしました．ボリュームシリアル番号に年月日を使用するなどで，メディアの管理に役立ちます．ただし，メディアの論理フォーマットを直接編集するので，ボリュームシリアル番号を編集する前にバッ

クアップをとっておき，障害が発生した場合に備えておくことをおすすめします．

③ ボリュームラベルの変更

ボリュームラベルの変更はドライブのプロパティからでも行えますが，単体でも行えるように実装しました．ドライブのバックアップ元メディアの識別が楽になると思います．

5）詳細設定など

① 基本設定

「基本設定」では，ドライブおよびメディアのプラグイン時に行うバックアップのタイプなどを指定できます(**図4**)．また，ファイルバックアップモードで，次に説明するファイル移動型バックアップもできます．

② 移動型バックアップ(フォルダ構造は保存)

ファイル移動型バックアップは，イメージバックアップと併用することで安心感が増しますが，自動的に元のファイルが削除されるので，注意して使用してください．

③ バックアップを保存するフォルダの変更

バックアップ先フォルダは，通常の場合，起動フォルダを基準として，初めに「BACKUP」フォルダが作成されます．そしてそのフォルダを標準フォルダとして使用します．バックアップ先を手動入力するか，もしくはブラウズ入力で設定した場合には，指定されたフォルダを基準としてバックアップが行われます．

④ 起動時最小化設定

「起動時最小化」チェックを行うことにより，Windowsの起動時にツールバーアイコンになります(**図5**)．この設定により，デバイスがプラグインされるまで隠しておくことができます．

⑤ ボリュームシリアル番号フィルタ

誤って大容量なFAT32ドライブをバックアップしてしまわないようにするために，ボリュームシリアル番号を登録し，登録されている番号以外のドライブをバックアップしないよう指定することができます(**図6**)．シリアルフィルタを有効にした場合，ドライブレターが同じになってしまう場合でも，ボリュームシリアルが登録されていなければバックアップは行いません．メディアの管理などに利用できると思います．

〔図4〕基本動作設定画面

〔図5〕起動したときのバルーンメッセージ表示

〔図6〕個別処理を行う画面

特殊なバックアップ＆フォーマッタとの組み合わせ

特殊な例として，Windows XP上でFAT32フォーマットを行ってしまった際，FAT16に戻しても認識しない場合があります．通常は，FAT16で物理フォーマットを行えば戻るはずなのですが，なぜか戻らないとき，まず購入時に生の情報をバックアップしておき，そのバックアップイメージでリカバリを行って，さらにWindows XPでFAT16フォーマットを行うことで，再び使用できるようになる場合もあるようです．古いデジカメなどでうまくいかない場合には，この方法を試してみてください．

メディアによっては，製造時の物理フォーマットと論理フォーマットが同じ場合，フォーマットイメージの使い回しができることもあるようです．筆者がテストした環境では，先に記したようなFAT32フォーマットを行ってしまって，デジカメでフォーマットができなくなったメディアが再び使用できるようになりました．

Plug&Backupをどのようにして行うのか

ソースコードとバイナリを公開する(`http://openlab.jp./kitaro/index-m.html`)ので，細かい動作については，

デバッグモードで実行して動作をトレースするか，ソースを追いかけてみてください．

多くの読者は，GUIなどの実装よりも，いかにして，

- 論理ドライブを読み取っているか
- デバイスのプラグイン情報を調べているか

という点を知りたいことでしょう．

また，さらに今回はInternet Explorer 5.0以降で拡張されたシェル機能を使用したタスクバー操作も実装してあります．

ここでは，これら三つの項目についてプログラムの解説を行います．

● デバイスのプラグインイベントを確認する

デバイスのプラグイン/プラグアウトを確認する方法として，従来ならばタイマを使用して一定時間ごとに確認する方法を用いることが多いと思います．しかし，この方法では一定時間ごとにシステムリソースが必要になってしまいます．そこで，デバイス変更の通知メッセージをフックすることにしました．

MFCを使用したVisual C++アプリケーションとして構築したので，デバイス変更の通知メッセージのフックは，次のようになります．

1）クラスの定義に，

```
afx_msg BOOL OnDeviceChange(UINT nEventType,
                             DWORD_PTR dwData );
```

を登録する

2）メッセージマップに，

```
ON_WM_DEVICECHANGE()
```

を追加する

3）`BOOL OnDeviceChange(UINT nEventType, DWORD_PTR dwData );` 関数を追加する

このようにして，デバイス変更にかかわるメッセージをフックします．

この方法を使用することで，特定のデバイスのプラグイン/プラグアウトを，アプリケーションレベルからサービスレベルまで幅広く利用できると思います．

● デバイスの論理フォーマットにアクセスする

今回はデバイスの論理フォーマットにアクセスするだけではなく，デバイス状態を確認してデバイスが有効になった場合に自動判定を行い，バックアップ動作までさせる設計になっています．そのため，単に論理デバイスにアクセスするのではなく，状態を一段階キャッシュするラッパクラスも作成しました．ラッパクラスでは，登録ドライブのデバイスに関する情報の取得を行い，キャッシュの内容と比較することで，デバイスが利用可能になったのかどうかを判定しています．

では，実際にWindows XPで論理ドライブの論理フォーマットへのアクセスはどのような手順で行うかを説明していきます．

● class DriveStatus，class LogicalDriveStatus

ドライブの物理状態を確かめるには，GetVolumeInformation関数を使用します．この関数は，

```
class DriveStatus : SetDrive
```

メソッドで行っているように，ドライブ名と\マークの組み合わせで指定する論理ドライブのボリューム情報を取得します．FAT/FAT32の場合は，この情報が物理的な割り当て情報になります．

なぜ，この情報が必要かというと，デバイスの論理的な読み取りと書き込みは，物理セクタのバイト数の整数倍でなければ失敗するためです．プログラムにおけるボリュームシリアル番号の書き込みでは，今回は便宜的に4096バイトなどを読み取り/書き込み時に指定していますが，本来ならセクタサイズから読み取り/書き込みバイト数を定義するのが正しい方法です．

次にデバイスへのアクセスですが，Windows XPなどではハンドルによってファイルのように操作できるため，まずCreateFileでハンドルを生成し，ReadFile/WriteFileを使用して読み書きを行います．ドライブの指定方法などがわかってしまえば通常のファイル操作と変わるところは少ないので，とくに悩むことはないでしょう．

例を**リスト1**に示します．

● シェル機能を使用する

ツールバーなどのシェル機能を実装するためには，マイクロソフトから配布されているPlatform SDKが必要になります．Platform SDKは，MSDNメンバに配布されているものなので，入手するためにはMSDNメンバになることが必要となります．

今回のプログラムで必要となるのは，おもにCore SDKやInternet Explorer SDKなどです．コンパイル時に，これらのヘッダ情報が必要になるのですが，Windowsのシェル拡張機能に関する情報が入手できるなど，通常のアプリケーションから一歩進めたUIを実装するためのヒントも多数収録されています．

さて，ヘッダやオンラインヘルプが収録されているとはいえ，そのままでは使用できません．そこで，使用しやすいようにヘルパークラスを作成してみました．

ここでは，シェルを呼び出すヘルパークラスのヘッダを紹介します(**リスト2**)．

このヘルパークラスで行っていることは，Shell32APIの一つである「NotifyIcon」を必要に応じて呼び出すことや，アニメーションアイコンをタイマで行うクラスの実装です．NotifyIcon関数では，アイコン関連の操作だけでなく，バルーンメッセージを表示する機能などももっているので，バックグラウンド処理を行わせるにはなかなか便利です．

● CListCtrlのヘルパークラス

メインプログラムの動作を助けるのに，今回はCListCtrlクラスを使用し，レポート形式で表示を行っています．CListCtrlクラスは，いくつかのコンビネーションで動作しているため，使い方がわりと複雑なのですが，比較的簡単になるように，

```
class CListCtrlSup
```

というクラスを作成しました．このクラスはソースコードに含まれているので参照してください．

〔リスト1〕デバイスへのアクセス

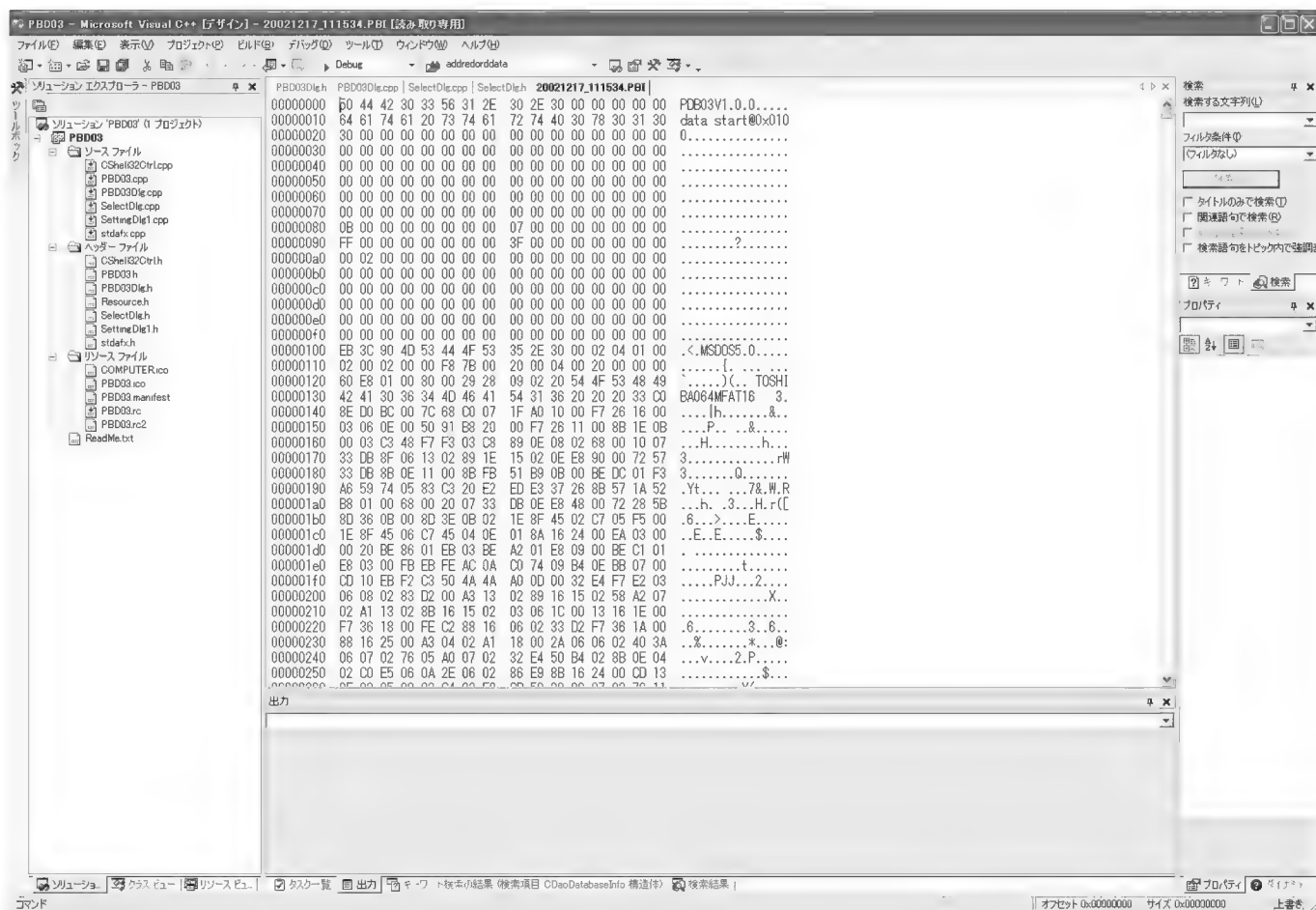
```
    char bufx[MAX PATH];
    HANDLE h
    wsprintf(bufx,"\\\\.\\%c:",  nDrive+'A');
    h = CreateFile(bufx,GENERIC_READ,0,NULL, OPEN_EXISTING,FILE_ATTRIBUTE_NORMAL, NULL);
    if((h!=NULL)&&(h!=(HANDLE)-1)){
         DWORD rt,rs;
         rs=4096;
         bWrite  = ::ReadFile(h,buf,rs,&rt,NULL);
         CloseHandle(h);
    }

class DriveStatus{
private:
    char sDrive[MAX PATH];  // drive root letter
    int nDrive;             // drive number
    // drive status
    char fsn[MAX PATH],vol[MAX PATH];
    DWORD vs,mcl,fsf;
    BOOL bResult;
    int nType;
public:
    int       GetType             (void){return nType;};
    BOOL GetLastError   (void){return bResult;};
    DWORD     GetFileSystemFlag(void){return fsf;};
    DWORD     GetMCL              (void){return mcl;};     // MaximumComponentLength
    DWORD     GetVolumeSerial     (void){return vs;};
    BOOL GetFSName      (CString &name){name=fsn; return bResult;};
    BOOL GetVLName      (CString &name){name=vol; return bResult;};
    BOOL ReadStatus          (void){
         nType = -1;    vs=mcl=fsf=0; ZeroMemory(fsn,MAX PATH); ZeroMemory(vol,MAX PATH);
         bResult = ::GetVolumeInformation(sDrive,vol,MAX PATH,&vs,&mcl,&fsf,fsn,MAX PATH);
         if(bResult ==TRUE) nType = ::GetDriveType(sDrive);
         return bResult;
    };
public:
    int SetDrive(int newDrive){  int nLastDrive = nDrive; nDrive = newDrive; wsprintf(sDrive,"%c:\\",nDrive+'A'); bResult=0;
                                                                                                return nLastDrive;   };
public:
    DriveStatus(int n){SetDrive(n);};
    DriveStatus(){SetDrive(0);};
    ~DriveStatus(){};
};

class LogicalDriveStatus{
private:
    DriveStatus *ds;
private:
    BOOL bLastStatus;
    DWORD     dwLastSerial;
    CString sLastVLName;
    CString sLastFSName;
    int       nLastType;
private:
    BOOL bCurStatus;
    DWORD     dwCurSerial;
    CString sCurVLName;
    CString sCurFSName;
    int       nCurType;
public:
    int       GetCurType(void){return nCurType;};
    BOOL GetCurError(void){return bCurStatus;};
    DWORD     GetCurSerial(void){return dwCurSerial;};
    BOOL GetCurVLName(CString &name){name=sCurVLName;return bCurStatus;};
    BOOL GetCurFSName(CString &name){name=sCurFSName;return bCurStatus;};
    BOOL IsChanged(void){
         BOOL bResult=FALSE;
         if(bLastStatus!=bCurStatus)bResult=TRUE;
         if(dwLastSerial!=dwCurSerial)bResult=TRUE;
//       if(nLastType!=nCurType)bResult=TRUE;
         if(strcmpi(sLastVLName,sCurVLName)!=0)bResult=TRUE;
         if(strcmpi(sLastFSName,sCurFSName)!=0)bResult=TRUE;
         return bResult;
    }
    BOOL UpdateStatus(void){
         bLastStatus    = bCurStatus;  dwLastSerial = dwCurSerial;  sLastVLName = sCurVLName;  sLastFSName = sCurFSName;
         nLastType = nCurType;
         dwCurSerial = -1; sCurVLName = ""; sCurFSName="";
         nCurType=-1;
         if((bCurStatus =ds->ReadStatus())!=FALSE){dwCurSerial=ds->GetVolumeSerial();ds->GetVLName(sCurVLName); ds
                                                                                              ->GetFSName(sCurFSName);}
         return bCurStatus;
    };
private:
    void InitialyzeFirst(int n){
         ds = new DriveStatus(n);
         dwLastSerial = -1; sLastVLName = ""; sLastFSName=""; nLastType=-1;
         if((bLastStatus =ds->ReadStatus())!=FALSE){ dwLastSerial=ds->GetVolumeSerial(); ds->GetVLName(sLastVLName); ds
```

〔リスト1〕デバイスへのアクセス(つづき)

```
                                                             ->GetFSName(sLastFSName); nLastType=ds->GetType(); }
        nCurType = nLastType;
        bCurStatus = bLastStatus; dwCurSerial = dwLastSerial; sCurVLName = sLastVLName; sCurFSName = sLastFSName;
    };
public:
    LogicalDriveStatus(int n){InitialyzeFirst(n);};
    LogicalDriveStatus(){InitialyzeFirst(0);};
    ~LogicalDriveStatus(){delete ds;};
};
```

〔リスト2〕ソースコード(CShell32Ctrl.h)

```
//==========================================================================
// Internet Explorer shell extention wrapper class for IE 5.0 or later
// copyright (c) kitaro 2000-2003
//
// before use this class... 'STDAFX.H" should custmize... see under
//
// #ifndef WINVER                  // Windows 95 および Windows NT 4 以降のバージョンに固有の機能の使用を許可します.
// #define WINVER 0x0500           // これを Windows 98 および Windows 2000 またはそれ以降のバージョン向けに適切な値に変更してください.
// #endif
//
// #ifndef _WIN32_WINNT            // Windows NT 4 以降のバージョンに固有の機能の使用を許可します.
// #define _WIN32_WINNT 0x0500     // これを Windows 98 および Windows 2000 またはそれ以降のバージョン向けに適切な値に変更してください.
// #endif
//
// #ifndef _WIN32_WINDOWS          // Windows 98 以降のバージョンに固有の機能の使用を許可します.
// #define _WIN32_WINDOWS 0x0490 // これを Windows Me またはそれ以降のバージョン向けに適切な値に変更してください.
// #endif
//
// #ifndef _WIN32_IE               // IE 4.0 以降のバージョンに固有の機能の使用を許可します.
// #define _WIN32_IE 0x0600        // これを IE 5.0  またはそれ以降のバージョン向けに適切な値に変更してください.
// #endif
//
// also if you want to use this class. you have to install Platform Core.SDK and Internet Explorer SDK
//

#ifndef __KITAROLIB_SHELL32CTRL__
#define __KITAROLIB_SHELL32CTRL__

#include <Shellapi.h>

#ifdef NOTIFYICONDATA_V1_SIZE

#define  KITAROLIB_SHELL32CTRL_USE

class CShellStatus {
    //============================================
    // Version 5.0
public:
    BOOL SetInfo(DWORD id,CWnd *pcWnd,LPSTR szTitle,LPSTR szInfo,DWORD dwTimeout = 10000,DWORD dwInfoOpt=NIIF_NONE);
                                                                                                  // ツールアイコンに吹き出し表示
    BOOL SetInfo(DWORD id,HWND hWnd,LPSTR szTitle,LPSTR szInfo,DWORD dwTimeout=10000,DWORD dwInfoOpt=NIIF_NONE);
                                                                                                  // ツールアイコンに吹き出し表示
    //============================================
private:
    static    BOOL NotifyIcon(DWORD dwMessage, PNOTIFYICONDATA lpdata);
public:
    static VOID CALLBACK CShellStatusTimerProf(HWND hwnd,UINT uMsg,UINT_PTR idEvent,DWORD dwTime);
    void AnimageIcon(DWORD iConID,HICON endIcon,HICON *phIcons,int pnIcons,int nTimer,CWnd *parentWnd);
    //============================================
private:
    BOOL bInit;
    BOOL bCanUse5;
    NOTIFYICONDATA nd;
public:
    static    void PumpMessage(void);
    void PumpOneMessage(void);
public:
    BOOL SetTIP         (DWORD id,HWND hWnd,LPSTR szTip);                              // TIPを設定
    BOOL AddIcon   (HICON hIcon,DWORD id,HWND hWnd,LPSTR szTip=NULL);                  // ツールバーにアイコン追加
    BOOL AddIcon   (HICON hIcon,DWORD id,CWnd *pcWnd,LPSTR szTip=NULL);                // ツールバーにアイコン追加
    BOOL DeleteIcon     (DWORD id,HWND hWnd);                                          // ツールバーからアイコン削除
    BOOL DeleteIcon     (DWORD id,CWnd *pcWnd);                                        // ツールバーからアイコン削除
    BOOL ModifyIcon     (HICON hIcon,DWORD id,HWND hWnd,LPSTR szTip=NULL);             // アイコン変更
    BOOL ModifyIcon     (HICON hIcon,DWORD id,CWnd *pcWnd,LPSTR szTip=NULL);           // アイコン変更
    BOOL SetCallback(DWORD callbackid,HWND hWnd);                                      // コールバックを設定
    BOOL SetCallback(DWORD callbackid,CWnd *pcWnd);                                    // コールバックを設定
public:
    CShellStatus();
    ~CShellStatus();
};
#endif

#endif // __KITAROLIB_SHELL32CTRL__
```

〔図7〕イメージバックアップされたファイルのダンプ表示

バックアップイメージファイルのフォーマットについて

バックアップイメージファイルは，論理フォーマット全体をバックアップしたものです．物理フォーマット構造にしたがってバックアップされるため，元の物理フォーマット情報（ドライブジオメトリ）をヘッダ情報として256バイトに保存し，その後，論理フォーマットデータを非圧縮で保存しています．

FAT12/FAT16/FAT32のイメージを仮想ドライブとして使用できるツールなどがある場合，ヘッダの256バイトを削除することで対応できるかもしれません．

Visual Studio.NETやバイナリエディタでイメージバックアップファイルを開くと**図7**のように，メディアディスクリプタテーブルなどまで表示されます．論理フォーマットの読み込みでは，ここまで読み込めるので，メディアに対して修復を行う前に一度バックアップをとっておくことで，修復によって逆にデータをロストしてしまうといった事故をふせぐことができるでしょう．このような例はまれですが，NTFSと異なりFATではファイル情報の破壊が比較的起こりやすいため，事前の予防策を講じておくことをおすすめします．

*　　　　　*

このように，ドライブの論理イメージそのものをバックアップすることで，ファイル全体を個別にバックアップするのではなく，構造全体をバックアップ&修復することが可能になります．もちろん，高度なバックアップ機能をもったツールも多数販売されていますが，高度な機能がなくとも，今回紹介したようなプログラムで簡単にバックアップできます．

ひろはた・ゆきお　OpenLab.

第16回

BASICのプログラムをCのプログラムに変換するコンバータ ―BCX

水野貴明

今回紹介するBCXは，BASICを使ってWindowsのアプリケーションを作成するためのツールだが，コンパイラでもインタプリタでもない．BCXは，BASICのプログラムをCのプログラムに変換するコンバータなのだ．BCXを使ってBASICのプログラムをCに変換し，それをCのコンパイラを使ってコンパイルすることで，アプリケーションを作成することができるというものである．

DATA

● **BCX**
作者：Kevin Diggins
Webサイト：`http://bcx.basicguru.com/`
現在のバージョン：3.00（2002年10月5日更新）

● **LCC-Win32**
作者：Jacob Navia
Webサイト：`http://www.cs.virginia.edu/~lcc-win32/`
現在のバージョン：3.8（2002年12月13日更新）

インストール

ダウンロードできるBCXのパッケージは，標準的なインストーラ形式になっており，簡単にインストールができる．BCX自体は単体のアプリケーションだが，パッケージ内容としては，プログラムを記述するための「JFE Code editor」，メニューやダイアログの生成を簡単に行うためのツール，サンプルプログラムなど，さまざまなファイルの集合体となっている（**表1**）．

また，BCXをインストールするにあたっては，利用するコンパイラであるLCC-Win32も同時にインストールしておくとよい．BCXは，あくまでBASICのプログラムをCに変換するだけのものなので，Cのコンパイラがないと，アプリケーションを作ることはできないからである．

もちろん，CのコンパイラはLCCだけではないのだが，BCXはフリーのコンパイラであるLCCを有効に活用するために開発されたもの，とのことで，LCCでコンパイルすることを前提としたコードを出力する．Borland C++ Compilerなど，ほかのコンパイラがインストールされており，それを利用するつもりであれば，それでもかまわないが，Cに変換したコードを修正しなければならない場合もある．

LCC-Win32の配布パッケージは，BCXと同様に一般的なインストーラ形式になっているので，ダウンロードしたインストーラを実行すればよい．途中で指定するのは，インストール先のファイルパスだけである．ファイルのコピーが終了すると，ライブラリのプレビルドが始まる．100個以上のライブラリをビルドするので，多少時間がかかる（**図1**）．

なお，LCCをあらかじめインストールしてからBCXをインストールすることで，BCXは変換やコンパイルを自動化したバッチファイルを自動的に更新してくれる．したがって，BCXよりも先に，LCCをインストールしておくとよいだろう．

二つのプログラムがインストールできたら，続いて簡単に実行できるようにするために，それぞれの実行プログラムのパスを追加する．パスの追加の仕方はOSによって異なるが，Windows 2000などであれば，「システムのプロパティ」の「詳細」タブの「環境変数」ボタンを押すことで，環境変数の追加ダイアログを表示し，`PATH`という名の環境変数にそれぞれの実行プログラムのパスを追加する．追加するパスは，以下のようにそれぞれのインストール先ディレクトリ下の「`bin`」ディレクトリである．

（BCXのインストールディレクトリ）`\bin\`
（LCCのインストールディレクトリ）`\bin\`

〔表1〕**BCXに含まれるツール**

BCX TO EXE Tool	GUI上でBCX，LCCを使ったEXEファイル作成をできるツール
Dialog Editor	ダイアログリソースを作成するツール
JFE Code Editor	ソースコードを記述するためのエディタ
Menu Maker	GUIアプリケーションのMENUをつくるコードを結果を見ながら作成できるツール
Windows MsgBox Creator	BCXの`MessageBox`関数で表示するダイアログを，簡単に生成するツール

〔図1〕LCCのライブラリがビルドされているところ

BASICプログラムの変換とコンパイル

BCXの基本的な使い方は，テキストエディタでBASICのプログラムを書き，BCXでCのプログラムへと変換する．そしてそれを，LCCなどのCのコンパイラでコンパイル，リンクを行って実行ファイルを作成する，という流れになる．

では，簡単なプログラムのCへの変換およびコンパイルを行いながら，その実際の使い方を見ていくことにしよう．用意したのは，次のようなプログラムである．

```
DIM a$
INPUT "--->";a$
PRINT a$
```

このプログラムは，コンソール上で動作するもので，画面上からデータを入力すると，それをそのまま表示するというプログラムである．これをたとえば「input.bas」という名前で保存していたとしたら，これをBCXでCのプログラムに変換するには，次のように指定する．

```
> bc input.bas
```

すると変換が行われ，「input.c」というファイルが生成される．変換後のCのファイルを**リスト1**に示す．文字列が2048バイトの配列として定義され，そこに入力データを読み込んで出力する，という先ほどのBASICのプログラムとほぼ等価なCのプログラムになっていることがわかる．

BCXの仕事はここまでである．続いて，これをLCCを使ってコンパイルする．

```
> lcc input.c
```

コンパイルが行われると「input.obj」というオブジェクトファイルが生成されるので，最後にリンカを使って実行用ファイルを生成する．

```
> lcclnk input.obj
```

これで，「input.exe」という実行用のアプリケーションファイルが作成される．

〔リスト1〕BCXの生成したCのコード

```
// *********************************************************
//   Created with BCX -- The BASIC To C Translator (ver 3.00)
//      BCX (c) 1999, 2000, 2001, 2002 by Kevin Diggins
// *********************************************************
#include <windows.h>    // Win32 Header File
#include <windowsx.h>   // Win32 Header File
#include <commctrl.h>   // Win32 Header File
#include <mmsystem.h>   // Win32 Header File
#include <shellapi.h>   // Win32 Header File
#include <shlobj.h>     // Win32 Header File
#include <winsock2.h>   // Win32 Header File
#include <richedit.h>   // Win32 Header File
#include <conio.h>
#include <ctype.h>
#include <math.h>
#include <stdio.h>
#include <string.h>
#include <stddef.h>
#include <stdlib.h>
#include <setjmp.h>
#include <time.h>
#include <stdarg.h>

// *************************************************
//           User Global Variables
// *************************************************

static char   a[2048];

// *************************************************
//                 Main Program
// *************************************************

int main(int argc, char *argv[])
{
printf("%s","--->");
gets(a);
printf("%s\n",a);
return 0;   //  End of main program
}
```

ここでは，コマンドを一つ一つ実行したが，BCXのbinディレクトリには，BCXによる変換とLCCによるコンパイル，リンクを行うためのバッチファイルが用意されている．これは「LCALL.BAT(コンソールプログラム用)」，「LDALL.BAT(DLL作成用)」，「LWALL.BAT(GUIアプリケーション用)」の三つである．

これらのバッチファイルを利用すると，BASICのプログラムを指定するだけで，簡単に実行用ファイルを作成できる．

```
> lcall input
```

BCXの言語体系

繰り返しになるが，BCXはBASICからCへの変換を行うツールである．ただし，どんなBASICプログラムでも変換が可能というわけではなく，BCXで定められた言語仕様にのっとって書かれたプログラムが変換できる，というものである．したがってBCXは，いってみれば単なるコンバータではあるが，一つのBASIC系プログラミング言語の開発環境である，という見方のほうが正しいように思う．

ヘルプファイルによると，BCXの言語仕様はQuickBasic，VisualBasic，PowerBasicを混ぜたもの，ということである．命令の一覧表を見ると「PRINT」，「INPUT」，「INKEY$」……とい

った，ほかのBASIC言語を使っていればおなじみの命令が並んでいる．そのほかにも，「.wav」サウンドファイルを実行するPLAYWAVや，ネットワーク経由でファイルのダウンロードを行うDOWNLOADなど，有用な命令が用意されている．

「BCX_」で始まる名前をもつBCXオリジナルの命令も存在する．BCXはコンソールアプリケーションだけでなく，WindowsのGUIアプリケーションも作成できるようになっている．「BCX_」で始まる名前をもつのは，ダイアログやボタン，チェックボックスといったGUIコントロールを作成するための関数や，画面上に線や矩形を書くための描画コマンドなどである．

また，BCXでは新しいプロシージャ，関数を定義することも可能である．プロシージャを定義するには「SUB～END SUB」，関数なら「FUNCTION～END FUNCTION」という宣言文を用いる．関数の場合は，戻り値は「FUNCTION = Z$」のように指定することになる（**リスト2**）．

BCXでは，利用する変数は必ず宣言しなければならない．宣言にはDIM文を用いる．変数の宣言は，旧来のBASICのように「$」や「%」を使って型を表すこともできるし，「AS INTEGER」のように指定してもよい．したがって，以下の二つの宣言文は同じ意味になる．

```
DIM a%
DIM a AS INTEGER
```

変数のスコープは，プロシージャ，関数内で宣言されている場合はローカルに，それ以外はグローバルになる．

さてもう一つ，BCXにはなかなかユニークな仕様が存在する．それは，内部にCのコードを直接埋め込み，BASICとCのハイブリッドコードを作ることができる点である．そもそもCに変換するツールであるから，それができても何の不思議もないのだが，たとえば既存のCの関数を自分のBASICのコードに埋め込む，といったことも簡単にできるのは，単なるBASICのコンパイラにはできない芸当である．

Cのコードを埋め込むには，Cのコードの両側を「$CCODE」というキーワードで囲むことで可能となる（**リスト3**）．また，1行だけを記述するのであれば，行頭に「!」をつけて記述することも可能である．

```
DIM i
i = 10
! i++;        // ←ここだけCのコード
PRINT i
```

BCXの変換の特徴

それでは続いて，BCXがどのようにC言語への変換を行うのか，という部分を見ていくことにする．BASICのプログラムと，それを変換したCのプログラムを比較してみると，その変換のスタンスは，標準ライブラリ関数で対応できる部分は対応し，それ以外は独自の関数を埋め込む，といった感じであることがわかる．たとえばPRINT文は，通常，以下のように変換される．

変換前：PRINT "ABC"

変換後：printf("%s","ABC");

これは，標準ライブラリ関数で対応している例である．それに対し絶対値を求めるABSは，Absという新しい関数が追加され，そこで処理が行われる（変換後のコードは**リスト4**）．

変換前：A = ABS(-10)

こういった独自の関数は，それらを利用するコマンドや関数を利用したときにのみ追加されるので，不必要なコードでコードサイズが増大する心配はない．

ただし，注意しなければならないのは，BCXにはエラーのチェックを行う機構がないということである．もし，BASICのプログラムに不備があった場合でも，とくにエラーを返すことなく，その部分を無視して変換を行ってしまうのである．

たとえば，PRINT文を間違えて「PRINF」と打ってしまった場合，以下のように変換されてしまう．

変換前：PRINF "ABC"

変換後：PRINF"ABC";

つまり，BCXは構文に間違いがあった場合，BCXが解釈できなかった部分の文字列を「そのまま」出力してしまうのである．結果，LCCでコンパイルを行った場合に，エラーが発生してしまうことになる．

また，多くのBASICの場合，あらかじめ定義していない変数であってもプログラム中で問題なく使えてしまう．しかし，BCXの場合にDIM文での変数定義を怠ると，Cのプログラムでも変

〔リスト2〕BCXで関数を定義する

```
DIM A$

A$ = Blurb$(65,66,67)
PRINT A$

FUNCTION Blurb$(A,B,C)
DIM Z$
 Z$ = CHR$(A) & CHR$(B) & CHR$(C)
 FUNCTION = Z$
END FUNCTION
```

〔リスト3〕BASICとCのコードを混ぜて記述する

```
PRINT "ここはBASICのコード"

$CCODE
int nNumber;
nNumber = 15;
printf("nNumber is equal to : %d\n", nNumber);
$CCODE

PRINT "ここもBASICのコード"
```

〔リスト4〕ABS(絶対値を求める関数)の変換

```
a = Abs(-10);

double Abs (double a)
{
 if(a<0) return -a;
        return  a;
}
```

〔リスト5〕GUIアプリケーションを作るためのプログラム

```
GUI  "Sample"

DIM MyForm    AS  CONTROL
DIM MyText    AS  CONTROL
DIM MyButton  AS  CONTROL

SUB FORMLOAD
MyForm   = BCX FORM    ("GUIサンプル" ,               0,   0, 100,  80)
MyText   = BCX EDIT    ("入力"        , MyForm, 101, 10,  15,  80,  20)
MyButton = BCX BUTTON  ("ボタン"      , MyForm, 102, 20,  50,  60,  20)
CENTER (MyForm)
SHOW   (MyForm)
END SUB

BEGIN EVENTS
 DIM ret!
 SELECT CASE CBMSG
  CASE WM COMMAND
'**********************
  IF CBCTL = 102 THEN
    ret = MSGBOX( BCX GET TEXT$(MyText) , "出力テスト", MB OK)
  END IF
  CASE WM CLOSE
'**********************
  IF MSGBOX("終了してもよろしいですか?", "終了", MB YESNO) = IDYES THEN
    DestroyWindow(MyForm)
  END IF
  EXIT FUNCTION
 END SELECT
END EVENTS
```

〔図2〕BCXで作成したGUIサンプル

数定義が記述されず，こちらもやはりエラー発生の原因となってしまう．

したがって，プログラミング時のタイプミスや変数の定義し忘れなど，単純なミスであっても，発覚するのがLCCでのコンパイル時になってしまう可能性があり，これはBCXの大きな問題点であるといえる．

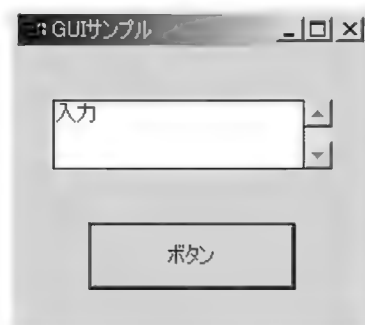
GUIプログラミング

BCXでは，GUIアプリケーションを作ることもできる．そこで，**リスト5**のような簡単なGUIプログラムを作成してみた．これは，**図2**のようなウィンドウを表示するプログラムである．テキストボックスとボタンが配置され，ボタンを押すとテキストボックスの内容がダイアログに表示される．そして，ウィンドウを閉じようとすると，終了してもよいかたずねるダイアログが表示される，というものだ．このプログラムを見ながら，BCXでのGUIプログラムの作成方法を見ていくことにする．

GUIアプリケーションを作成するためには，まずプログラムの先頭でGUI文を使って，プログラムがGUIを利用することを宣言する．

```
GUI  "Sample"
```

パラメータとしてはクラス名を指定する．これは，アプリケーションを識別するための文字列である．

そして，続いて利用するコントロールを格納する変数を定義する．コントロールを格納する変数の変数型は「CONTROL」を指定する．この変数型はCに変換した際にはHWND(ウィンドウハンドル)型として定義される．

```
DIM MyForm     AS  CONTROL
DIM MyText     AS  CONTROL
DIM MyButton   AS  CONTROL
```

さて，続いて実際のプログラム部分に入るわけだが，BCXでGUIプログラムを作るにあたっては，必ず定義しなければならないものが二つある．初期化を行うプロシージャとイベントハンドラである．

初期化を行うプロシージャは，「FORMLOAD」という名前で定義する必要がある．このプロシージャは，イベントループに入る前に呼び出される．ここでは通常ウィンドウやコントロールの配置を指定したり，データの初期化を行ったり，といった作業を行うことになる．**リスト5**の場合は，ウィンドウとテキストボックス，ボタンをそれぞれ配置して，ウィンドウを画面中央に移動(CENTER文)してから，表示(SHOW文)するという作業を行っている．

もう一つ定義しなければならないイベントハンドラとは，ボタンがクリックされた，テキストが入力された，といったイベントが発生したときに呼ばれるプログラムである．GUIプログラムの基本は，このイベントハンドラで発生したイベントをつかまえ，イベントに応じて処理を振り分けていく，というものになる．

呼び出された際，CBMSGという変数にイベントの名前が，CBCTLという変数にはイベントが発生したコントロールの番号が入る．また，CBWPARAM，およびCBLPARAMというパラメータから，イベントに関する情報を得ることができる．

イベントハンドラでは，イベントの種類とコントロールの番号によって，処理を振り分けていく必要がある．サンプルプログラムでは，ボタンが押されたときおよびウィンドウが閉じられたときに処理を行うようにしている．

ただし，どういったイベントがあるのかといったことは，BCXのヘルプファイルでは解説されていないので，自分で調べる必要がある．イベントの種類や行わなければならない処理については，CやC++など他の言語向けに書かれた解説であっても，まったく同じなので，BASIC向けに書かれたものでなくても

Windowsアプリケーション作成に関する書籍やWebページなどが役立つはずである．

また，GUIのアプリケーションを作る場合は，処理はすべて関数/プロシージャを作成してそこに記述しておく必要がある．コンソールアプリケーションの場合は，関数/プロシージャ外に書かれたコードはすべてmain関数内に置かれたが，GUIプログラミングの場合は，関数/プロシージャ外に書かれたコードはWinMainには入ってくれず，そのままCでも関数外に置かれてしまうため，LCCでのコンパイル時にエラーが発生してしまうからだ．

最後に，GUIアプリケーションのコンパイル方法だが，BCXのbinディレクトリに置かれている「LWALL.BAT」というバッチファイルを使うと簡単である．

```
> lwall gui_sample
```

日本語は使えるのか？

こういった海外製の開発環境を利用する場合，日本語がきちんと使えるかどうかというのは，非常に重要なポイントとなる．そこで，BCXで作成したプログラム中で日本語を使ってみる実験を行った．なお，検証作業はWindows2000上で行い，利用した日本語の文字コードはすべてシフトJISである．

まずは，PRINT文で日本語を表示するプログラムを作成してみる．変換結果のプログラムは，次のようになった．

変換前： `PRINT "あいうえお"`

変換後： `printf("%s","あいうえお");`

変換は問題なく行われているようである．このことから，BCX自体は，日本語がデータ中に含まれていても，問題なく変換を行ってくれることがわかった．さらに，これをコンソールアプリケーションとしてコンパイルして実行してみると，正しく「あいうえお」が表示されたので，LCCも問題なく日本語が含まれるプログラムをコンパイルできるようだ．

続いて，GUIを用いたサンプルで日本語が使えるかどうかを実験した．実験に用いたのは，先ほどGUIプログラムのサンプルとして用いたプログラム(**リスト5**)である．このプログラムでは，意識的に日本語を使うようにしたのだが，ウィンドウタイトル，テキストボックス，ボタン，メッセージボックス，とすべて正しく日本語で表示されている．また，テキストボックスではインライン入力(日本語の変換作業もすべてテキストボックスで行うこと)も可能となっており，日本語の利用には何の問題もないといえる．

最後に，日本語の文字のうち，2文字目の文字コードが「5Ch」になっている文字を利用した場合のチェックを行った．このコードは「¥」を表す文字コードで，Cの文字列中ではエスケープ文字として利用されるものである．

日本語非対応の開発環境では，このようにエスケープ文字が2バイト目に含まれるコードを利用すると不具合を生じるものが多いのだ．そこで，「噂(895Ch)」という漢字を表示してみたところ，問題なく表示が行われた．そこでCへ変換されたコードを見ると，以下のようになっていた．

変換前： `A$ = "噂"`

変換後： `strcpy(a,"噂\");`

つまり，「¥」というデータのあとにもう一つ「¥」をつけてあるのだ．Cの文字列では「¥¥」は「¥」を表すエスケープシーケンスなので，画面上には正しく「噂」と表示されるわけである．これはおそらく，BASIC中で普通に「¥」を使った場合に，それがエスケープ文字と判断されるのを防ぐための処理だと思われるが，結果として日本語も正しく表示できるようになっている．

これらの検証を見るかぎり，BCXでの日本語の使用はそれほど問題がないように見える．これまで本連載で紹介してきた外国産の開発環境は，日本語の扱いに難があったものが多かっただけに，これはなかなかうれしいことである．

おわりに

BASICでWindowsアプリケーションを作ることができるツールであるBCXを見てきたが，いかがだっただろうか．BASICとはいってもインタプリタではなく，Cに変換してコンパイルしてしまうので，完全なネイティブコードを生成できるのもなかなか魅力的であると筆者は感じた．

しかしBCXは，Cの知識がまったくない場合，利用するのは難しいのではないか，とも思う．その理由は，BCX自体のエラーを検知する機能がない点がある．したがって，エラーの検知はLCCでのコンパイル時に，LCCが出力した(C言語での)エラー情報を元に行う必要があるのだ．

また，GUIアプリケーションを作る場合，イベントハンドラ部分を作成するにあたっては，Windowsのイベント関連の知識も必要となる．しかも，このあたりの情報はヘルプにもあまり載っておらず，自分で調べなければならない．

これらのことから，BCXはBASIC以外の言語の知識がなかったり，BCXではじめてプログラミングを勉強しようとする，といった人にはまだ少し厳しい面があるように感じる．では，どんな人に向いているのかというと，C言語がある程度わかるが，Cですべてを書くのは面倒だと思っている人ではないかと思うのだ．

BCXはBASICで記述したプログラムを元に，必要なヘッダ情報やコードを追加してくれるので，プログラミング中にありがちな「定型作業」をかなり軽減してくれる．また，BASICのほうが記述しやすい部分はBASICで，Cのほうが記述しやすい部分はCで，といった書き方も可能になるので，非常に効率的なプログラミングができるのではないだろうか．

みずの・たかあき

x86CPUだけでもマスタしたい

開発技術者のためのアセンブラ入門

第⑯回　2進演算命令の乗除算と10進演算命令　大貫広幸

今回は前回の続きとして，2進演算命令の乗除算の命令と10進演算命令について説明します．

2進演算命令の乗除算命令

乗除算命令には，加減算命令とは異なり，扱う値の符号のある/なしにより使用する命令が異なります．

符号なし乗算はMUL命令，符号付き乗算はIMUL命令を使用します．そして，符号なし除算はDIV命令，符号付き除算はIDIV命令を使用することになります(**表1**)．

実際のMASMでの2進演算命令の乗除算命令の記述例を**リスト1**に，gasでの記述例を**リスト2**(p.148)に示します．

● MUL命令

扱う値に符号がない場合の乗算には，このMUL命令を使用します．

MUL命令は，転送先となるレジスタを固定としています．また，転送元となる二つの値のうち，一つをアキュムレータ(AL, AX, EAX)固定とし，もう一つをオペランドで指定します．

オペランドには，汎用レジスタかメモリ上の値を指定します．オペランドで指定される転送元をSOUとした場合，MUL命令は，オペランドサイズにより次のように演算されます．

▶オペランドSOUのサイズがバイトなら

AX ← AL × SOU

▶オペランドSOUのサイズがワードなら

DX:AX ← AX × SOU

▶オペランドSOUのサイズがダブルワードなら

EDX:EAX ← EAX × SOU

このように，乗算の結果(積)がワード以上の値になる場合は，二つのレジスタ(E)DXと(E)AXに分けて結果が格納されます．レジスタ(E)DXには積の上位ビット，レジスタ(E)AXには積の

〔**表1**〕
x86系の32ビットCPUで使用できる2進乗除算命令

分類	インストラクション名	動作	影響を受けるフラグ					
			OF	SF	ZF	AF	PF	CF
2進算術命令	MUL	Unsigned Multiply 2進数による符号なし乗算 AX ← AL × byteSOU DX：AX ← AX × wordSOU EDX：EAX ← EAX × dwordSOU	*	?	?	?	?	*
	IMUL	Signed Multiply 2進数による符号付き乗算 AX ← AL × byteSOU DX：AX ← AX × wordSOU EDX：EAX ← EAX × dwordSOU DEST ← DEST × SOU DEST ← SOU × IMM	*	?	?	?	?	*
	DIV	Unsigned Divide 2進数による符号なし除算 AX ÷ byteSOU →（商→AL，剰余→AH） (DX：AX) ÷ wordSOU →（商→AX，剰余→DX） (EDX：EAX) ÷ dwordSOU →（商→EAX，剰余→EDX）	?	?	?	?	?	?
	IDIV	Signed Divide 2進数による符号付き除算 AX ÷ byteSOU →（商→AL，剰余→AH） (DX：AX) ÷ wordSOU →（商→AX，剰余→DX） (EDX：EAX) ÷ dwordSOU →（商→EAX，剰余→EDX）	?	?	?	?	?	?

注1：表中のDESTはdestination(先)，SOUはsource(元)，byteSOUはサイズがバイトのsource，wordSOUはサイズがワードのsource，dwordSOUはサイズがダブルワードのsource，IMMはimmediate(定数)を表す
注2：表中の影響を受けるフラグの記号は次の状態を表す
？＝未定義　＊＝結果にしたがい変化する

下位ビットが，それぞれ格納されます(**図1**，p.149)．

乗算の結果，積の上位半分のビットがゼロならOFとCFのフラグが0にクリアされ，ゼロ以外の値になればOFとCFのフラグが1にセットされます．つまり，乗算の結果，積がオペランドで指定されたサイズのビットをオーバするとOFとCFのフラグが1になるわけです．これにより，乗算の結果がオーバフローしたか否かをフラグで知ることができます．

MUL命令では，OFとCF以外のステータスフラグ(SF，ZF，AF，PF)の設定は未定義となっています．そのため，MUL命令実行後のSF，ZF，AF，PFのフラグは使用できません．

〔リスト1〕MASMのMUL，IMUL，DIV，IDIV命令の記述例

```
                                  .586
                                  .model flat

00000000                          .data

00000000 01                       dtByte  db      1
00000001 0002                     dtWord  dw      2
00000003 00000004                 dtDWord dd      4

00000000                          .code
                                  ; MUL
00000000  F6 E1                           mul     cl
00000002  66| F7 E1                       mul     cx
00000005  F7 E1                           mul     ecx
00000007  F6 25 00000000 R                mul     dtByte
0000000D  66| F7 25                       mul     dtWord
          00000001 R
00000014  F7 25 00000003 R                mul     dtDWord
                                  ; IMUL
0000001A  F6 E9                           imul    cl
0000001C  66| F7 E9                       imul    cx
0000001F  F7 E9                           imul    ecx
00000021  F6 2D 00000000 R                imul    dtByte
00000027  66| F7 2D                       imul    dtWord
          00000001 R
0000002E  F7 2D 00000003 R                imul    dtDWord

00000034  66| 0F AF CE                    imul    cx,si
00000038  0F AF CE                        imul    ecx,esi
0000003B  66| 0F AF 0D                    imul    cx,dtWord
          00000001 R
00000043  0F AF 0D                        imul    ecx,dtDWord
          00000003 R
0000004A  66| 6B C9 12                    imul    cx,12h
0000004E  6B C9 12                        imul    ecx,12h
00000051  66| 69 C9 1234                  imul    cx,1234h
00000056  69 C9 12345678                  imul    ecx,12345678h

0000005C  66| 6B CE 12                    imul    cx,si,12h
00000060  6B CE 12                        imul    ecx,esi,12h
00000063  66| 6B 0D                       imul    cx,dtWord,12h
          00000001 R 12
0000006B  6B 0D 00000003 R                imul    ecx,dtDWord,12h
          12

00000072  66| 69 CE 1234                  imul    cx,si,1234h
00000077  66| 69 0D                       imul    cx,dtWord,1234h
          00000001 R
          1234

00000080  69 CE 12345678                  imul    ecx,esi,12345678h
00000086  69 0D 00000003 R                imul    ecx,dtDWord,12345678h
          12345678
                                  ; DIV
00000090  F6 F1                           div     cl
00000092  66| F7 F1                       div     cx
00000095  F7 F1                           div     ecx
00000097  F6 35 00000000 R                div     dtByte
0000009D  66| F7 35                       div     dtWord
          00000001 R
000000A4  F7 35 00000003 R                div     dtDWord
                                  ; IDIV
000000AA  F6 F9                           idiv    cl
000000AC  66| F7 F9                       idiv    cx
000000AF  F7 F9                           idiv    ecx
000000B1  F6 3D 00000000 R                idiv    dtByte
000000B7  66| F7 3D                       idiv    dtWord
          00000001 R
000000BE  F7 3D 00000003 R                idiv    dtDWord

                                  end
```

1オペランド形式のIMUL命令
IMUL　sou

2オペランド形式のIMUL命令
IMUL　dest，sou

3オペランド形式のIMUL命令
IMUL　dest，sou，imm

〔リスト2〕gasのMUL，IMUL，DIV，IDIV命令の記述例

```
 1                    .data
 2
 3  0000 01           dtByte:   .byte  1
 4  0001 0200         dtWord:   .word  2
 5  0003 04000000     dtDWord:  .long  4
 6
 7                    .text
 8                    # MUL
 9  0000 F6E1                 mulb    %cl
10  0002 66F7E1               mulw    %cx
11  0005 F7E1                 mull    %ecx
12  0007 F6250000             mulb    dtByte
12       0000
13  000d 66F72501             mulw    dtWord
13       000000
14  0014 F7250300             mull    dtDWord
14       0000
15                    # IMUL
16  001a F6E9                 imulb   %cl
17  001c 66F7E9               imulw   %cx
18  001f F7E9                 imull   %ecx
19  0021 F62D0000             imulb   dtByte
19       0000
20  0027 66F72D01             imulw   dtWord
20       000000
21  002e F72D0300             imull   dtDWord
21       0000
22
23  0034 660FAFCE             imulw   %si,%cx
24  0038 0FAFCE               imull   %esi,%ecx
25  003b 660FAF0D             imulw   dtWord,%cx
25       01000000
26  0043 0FAF0D03             imull   dtDWord,%ecx
26       000000
27  004a 666BC912             imulw   $0x12,%cx
28  004e 6BC912               imull   $0x12,%ecx
29  0051 6669C934             imulw   $0x1234,%cx
29       12
30  0056 69C97856             imull   $0x12345678,%ecx
30       3412
31
32  005c 666BCE12             imulw   $0x12,%si,%cx
33  0060 6BCE12               imull   $0x12,%esi,%ecx
34  0063 666B0D01             imulw   $0x12,dtWord,%cx
34       00000012
35  006b 6B0D0300             imull   $0x12,dtDWord,%ecx
35       000012
36
37  0072 6669CE34             imulw   $0x1234,%si,%cx
37       12
38  0077 66690D01             imulw   $0x1234,dtWord,%cx
38       00000034
38       12
39
40  0080 69CE7856             imull   $0x12345678,%esi,%ecx
40       3412
41  0086 690D0300             imull   $0x12345678,dtDWord,%ecx
41       00007856
41       3412
42                    # DIV
43  0090 F6F1                 divb    %cl
44  0092 66F7F1               divw    %cx
45  0095 F7F1                 divl    %ecx
46  0097 F6350000             divb    dtByte
46       0000
47  009d 66F73501             divw    dtWord
47       000000
48  00a4 F7350300             divl    dtDWord
48       0000
49                    # IDIV
50  00aa F6F9                 idivb   %cl
51  00ac 66F7F9               idivw   %cx
52  00af F7F9                 idivl   %ecx
53  00b1 F63D0000             idivb   dtByte
53       0000
54  00b7 66F73D01             idivw   dtWord
54       000000
55  00be F73D0300             idivl   dtDWord
55       0000
```

1オペランド形式のIMUL命令

IMUL {b, w, l} sou

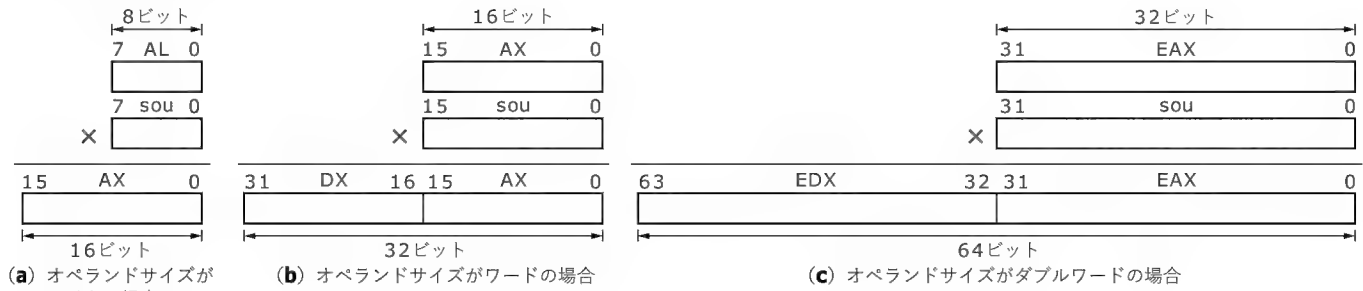
2オペランド形式のIMUL命令

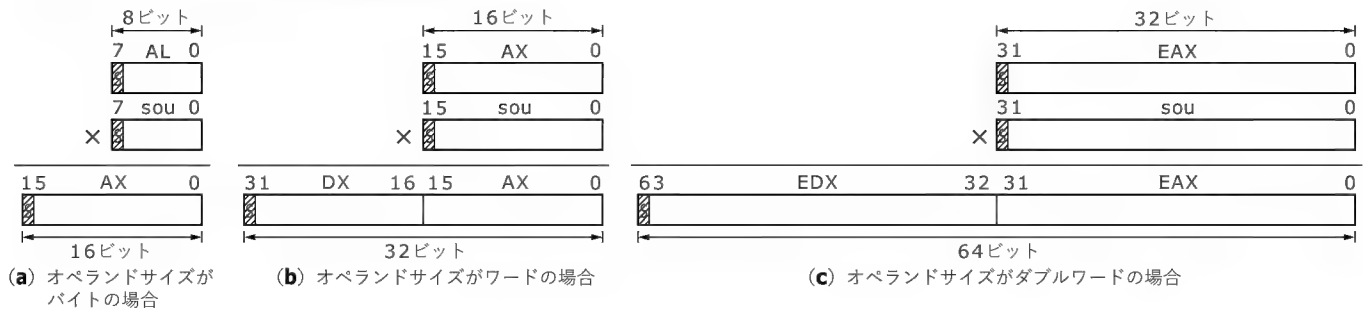
IMUL {w, l} sou，dest

3オペランド形式のIMUL命令

IMUL {w, l} imm，sou，dest

〔図1〕MUL命令の動作(MUL sou)

(a) オペランドサイズがバイトの場合　(b) オペランドサイズがワードの場合　(c) オペランドサイズがダブルワードの場合

〔図2〕1オペランド形式のIMUL命令の動作(IMUL sou)

(a) オペランドサイズがバイトの場合　(b) オペランドサイズがワードの場合　(c) オペランドサイズがダブルワードの場合

〔表2〕IMUL命令のバリエーション

形　式	演算内容
1オペランド形式 IMUL SOU	転送元となるレジスタあるいはメモリ，一つをオペランドとして指定する形式．指定オペランドのサイズにより次の演算が行われる ●オペランドサイズがバイト　AX ← AL × SOU ●オペランドサイズがワード　DX:AX ← AX × SOU ●オペランドサイズがダブルワード　EDX:EAX ← EAX × SOU
2オペランド形式 IMUL DEST, SOU	転送先となるレジスタと，転送元となるレジスタあるいはメモリ，イミディエイトの二つのオペランドを指定する形式 DEST ← DEST × SOU
3オペランド形式 IMUL DEST, SOU, IMM	転送先となるレジスタと，転送元となるレジスタあるいはメモリ，そして転送元に掛けられるイミディエイトの三つのオペランドを指定する形式 DEST ← SOU × IMM

注：表中のDESTはdestination(先)，SOUはsource(元)，IMMはimmediate(定数)を表す

● IMUL命令

符号付き整数の乗算には，IMUL命令を使用します．IMUL命令は，MUL命令に比べ命令のバリエーションが多く，**表2**に示すような「1オペランド形式」,「2オペランド形式」,「3オペランド形式」の3種類があります．

(1) 1オペランド形式のIMUL命令

1オペランド形式は，今述べたMUL命令を符号付き整数の乗算にしたもので，8ビット×8ビット，16ビット×16ビット，そして32ビット×32ビットの3種類の乗算が行えます．また，使用されるレジスタもMUL命令と同じになっています(**図2**)．

(2) 2オペランド形式のIMUL命令

2オペランド形式は，転送元と転送先をオペランドで指定するもので，16ビット×16ビットと32ビット×32ビットの乗算が行えます．この2オペランド形式のIMUL命令は，転送先をDEST，転送元をSOUで表した場合，

DEST ← DEST × SOU

と演算します．転送先(DEST)には16ビットあるいは32ビットの汎用レジスタを指定し，転送元(SOU)には，転送先(DEST)と同じサイズの汎用レジスタかメモリ上の値，あるいはイミディエイトが指定できます(**図3**)．

(3) 3オペランド形式のIMUL命令

3オペランド形式は，二つの転送元と一つの転送先をオペランドで指定するもので，16ビット×16ビットと32ビット×32ビットの乗算が行えます．この3オペランド形式のIMUL命令は，転送先をDEST，転送元1をSOU1，転送元2をSOU2で表した場合，

DEST ← SOU1 × SOU2

と演算します．転送先(DEST)には16ビットあるいは32ビットの汎用レジスタを指定し，転送元1(SOU1)には，転送先(DEST)と同じサイズの汎用レジスタやメモリ上の値を指定します．そして，転送元2(SOU2)にはイミディエイトを指定します(**図4**)．

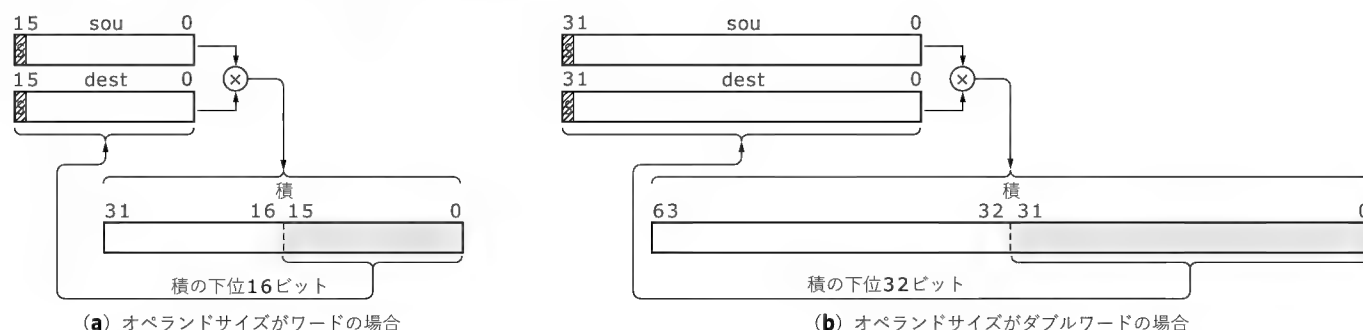
この3オペランド形式のIMUL命令は，MASMでは，

```
IMUL    dest, sou1, sou2
```

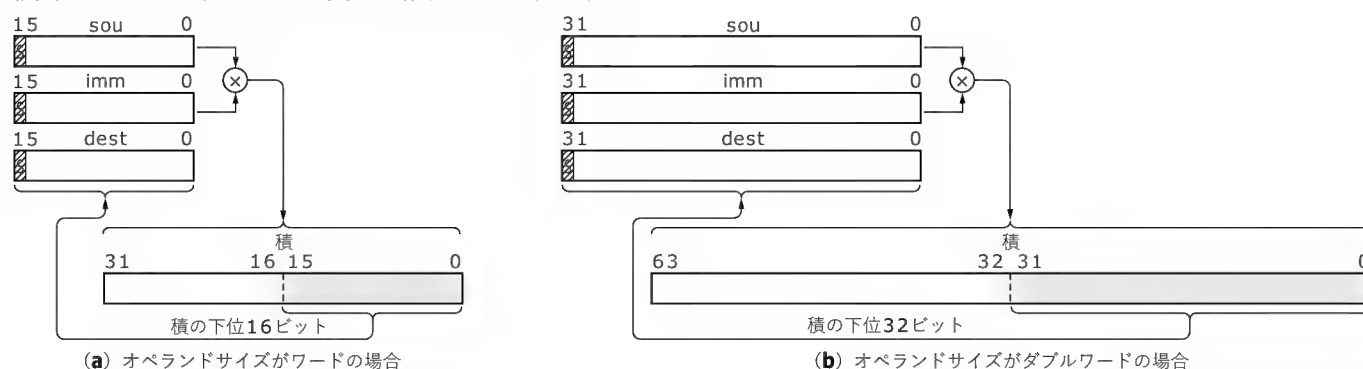
と記述しますが，gasでは，

〔図3〕**2オペランド形式のIMUL命令の動作**(IMUL dest, sou)

(a) オペランドサイズがワードの場合

(b) オペランドサイズがダブルワードの場合

〔図4〕**3オペランド形式のIMUL命令の動作**(IMUL dest, sou, imm)

(a) オペランドサイズがワードの場合

(b) オペランドサイズがダブルワードの場合

```
IMUL    sou2, sou1, dest
```

と記述することになります.

(4) 演算の結果

1オペランド形式の演算結果の積は，オペランドサイズの倍のビット数のレジスタに格納されましたが，2オペランド形式および3オペランド形式の演算結果の積は，オペランドサイズと同じビット数のレジスタに格納されます．そのため，2オペランド形式および3オペランド形式の場合，オペランドサイズ内に納まらない積は，下位のオペランドサイズのみが使用され，オーバした上位ビットは捨てられることになります．

このことから，1オペランド形式で得られた積は，常に正確な値となりますが，2オペランド形式および3オペランド形式で得られた積は，オペランドサイズ内に納まる値のみ正確な値となります．この点に注意が必要です．

フラグも1オペランド形式とそれ以外では，設定のされ方が多少異なります．

1オペランド形式の場合は，MUL命令と同じで積の上位半分のビットがゼロならOFとCFのフラグが0にクリアされ，ゼロ以外の値になればOFとCFのフラグが1にセットされます．

2オペランド形式および3オペランド形式では，転送先(DEST)に積が正確に格納された場合には，OFとCFのフラグが0にクリアされます．しかし，転送先(DEST)に積が収まり切れず，積の上位ビットが捨てられた場合には，OFとCFのフラグが1にセットされます．

IMUL命令では，OFとCF以外のステータスフラグ(SF, ZF, AF, PF)の設定は未定義となっています．そのため，IMUL命令の場合も，命令実行後のSF, ZF, AF, PFのフラグは使用できません．

● DIV命令

除算の命令の場合も，扱う値が符号なしか符号付きかで使用する命令が異なります．符号なし整数の場合はDIV命令を使用します．

DIV命令では，演算結果として商と剰余の二つの値が得られます．つまり，演算結果を格納する転送先が二つあるわけです．この転送先二つは固定されたレジスタに決められています．また，転送元となる二つの値のうち，被除数となる値を格納しているレジスタも固定されています．そのため，DIV命令は除数の値のみをオペランドとして指定します．オペランドは，汎用レジスタかメモリ上の値を指定します．

オペランドで指定される転送元をSOUとした場合，DIV命令は，オペランドサイズにより次のように演算されます．

▶オペランドSOUのサイズがバイトなら

AL ← (AX ÷ SOU)の商
AH ← (AX ÷ SOU)の剰余

▶オペランドSOUのサイズがワードなら

AX ← (DX:AX ÷ SOU)の商
DX ← (DX:AX ÷ SOU)の剰余

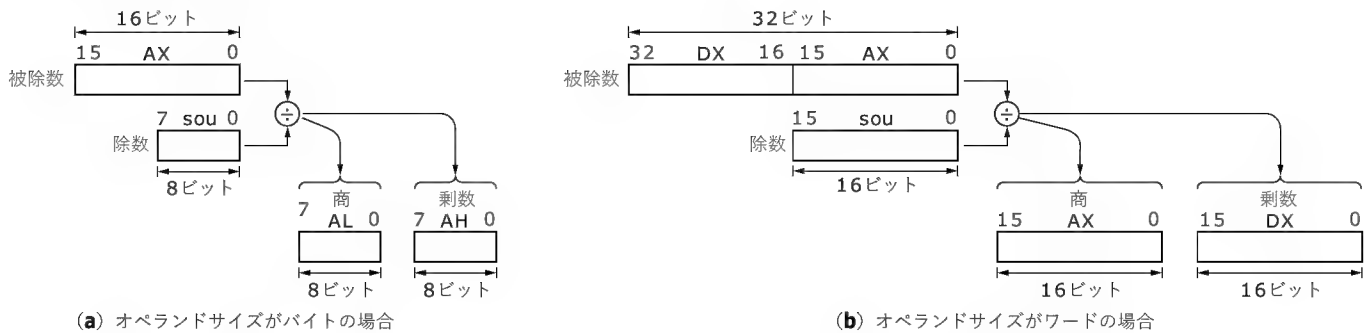
▶オペランドSOUのサイズがダブルワードなら

〔**図5**〕**DIV命令の動作**（DIV sou）

（**a**）オペランドサイズがバイトの場合

32ビット
32 DX 16 15 AX 0
被除数
15 sou 0
除数
16ビット
商
15 AX 0
剰余
15 DX 0
16ビット
16ビット

（**b**）オペランドサイズがワードの場合

（**c**）オペランドサイズがダブルワードの場合

EAX ← (EDX:EAX ÷ SOU)の商

EDX ← (EDX:EAX ÷ SOU)の剰余

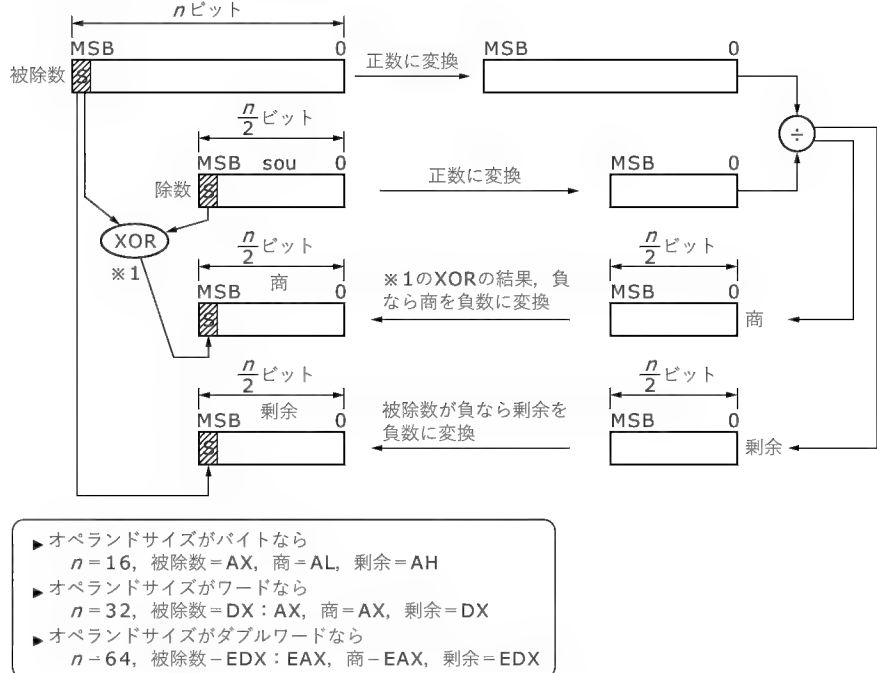
このように除算では，被除数がワード以上の値になる場合，二つのレジスタ(E)DXと(E)AXに分けて値を指定します．レジスタ(E)DXには被除数の上位ビット，レジスタ(E)AXには被除数の下位ビットを，それぞれ格納します(**図5**)．除算の結果，商が非整数となる場合は，0方向に切り捨てられた値が商となります．また，剰余は常に除数より小さな値となります．

DIV命令の場合，オペランドの除数がゼロの場合や商が指定レジスタに格納できない(オーバフローした)場合，フラグの設定ではなく，ベクタ番号0の例外発生となるので注意が必要です．例外が発生すると割り込みベクタ番号0の例外処理ルーチンにジャンプすることになります．そのため，この例外を発生させたくない場合は，DIV命令を実行する前に，事前に被除数および除数を調べておく必要があります．

〔**図6**〕**IDIV命令の動作**（IDIV sou）

DIV命令では，すべてのステータスフラグ(OF，SF，ZF，AF，PF，CF)の設定は未定義となっています．そのため，DIV命令実行後のステータスフラグは使用できません．

● IDIV命令

符号付き整数の除算には，IDIV命令を使用します．このIDIV命令は，今述べたDIV命令を符号付き除算にしたものといえます．そのため，演算の種類や使用されるレジスタといったものは，DIV命令と同じになっています．

IDIV命令の除算の結果，商が非整数となる場合は，0方向に切り捨てられた値が商となります．また，剰余は常に除数より絶対値が小さな値となり，被除数と同じ符号が付けられます(**図6**)．IDIV命令の場合，オペランドの除数がゼロの場合や商が符

〔表3〕x86系の32ビットCPUで使用できる10進算術命令

分類	インストラクション名	動　作	影響を受けるフラグ					
			OF	SF	ZF	AF	PF	CF
10進算術命令	DAA	Decimal Adjust after Addition パックBCDを2進加算した後に補正し，加算結果をパックBCDにする	?	*	*	*	*	*
	DAS	Decimal Adjust after Subtraction パックBCDを2進減算した後に補正し，減算結果をパックBCDにする	?	*	*	*	*	*
	AAA	ASCII Adjust After Addition アンパックBCDを2進加算した後に補正し，加算結果をアンパックBCDにする	?	?	?	*	?	*
	AAS	ASCII Adjust After Subtraction アンパックBCDを2進減算した後に補正し，減算結果をアンパックBCDにする	?	?	?	*	?	*
	AAM	ASCII Adjust After Multiply アンパックBCDを2進乗算した後に補正し，積を2桁のアンパックBCDにする	?	*	*	?	*	?
	AAD	ASCII Adjust Before Division 除算の被除数となる2桁のアンパックBCDを2進除算の前に2進整数に変換する	?	*	*	?	*	?

注：表中の影響を受けるフラグの記号は次の状態を表す
　？＝未定義　＊＝結果にしたがい変化する

〔リスト3〕MASMのDAA，DAS，AAA，AAS，AAM，AAD命令の記述例

```
                              .586
                              .model flat                      ┌── 左の10進演算を行った場合の結果を示す
00000000                      .code
00000000  B0 12                    mov     al,12h
00000002  04 34                    add     al,34h
00000004  27                       daa                          ; CF=0, AL=46h
                              ;
00000005  B0 98                    mov     al,98h
00000007  04 59                    add     al,59h
00000009  27                       daa                          ; CF=1, AL=57h
                              ;-------
0000000A  B0 34                    mov     al,34h
0000000C  2C 12                    sub     al,12h
0000000E  2F                       das                          ; CF=0, AL=22h
                              ;
0000000F  B0 12                    mov     al,12h
00000011  2C 38                    sub     al,38h
00000013  2F                       das                          ; CF=1, AL=74h=-26h (100-74=26)
                              ;-------
00000014  B4 00                    mov     ah,0
00000016  B0 03                    mov     al,03h
00000018  04 06                    add     al,06h
0000001A  37                       aaa                          ; CF=0, AX=0009h
                              ;
0000001B  B4 00                    mov     ah,0
0000001D  B0 08                    mov     al,08h
0000001F  04 06                    add     al,06h
00000021  37                       aaa                          ; CF=1, AX=0104h      減算の結果，
                              ;-------                                              負になる場合の値
00000022  B4 00                    mov     ah,0
00000024  B0 07                    mov     al,07h
00000026  2C 05                    sub     al,05h
00000028  3F                       aas                          ; CF=0, AX=0002h
                              ;
00000029  B4 00                    mov     ah,0
0000002B  B0 05                    mov     al,05h
0000002D  2C 07                    sub     al,07h
0000002F  3F                       aas                          ; CF=1, AX=FF08h=-02h (10-8=2)
                              ;-------
00000030  B0 03                    mov     al,03h
00000032  B1 07                    mov     cl,07h
00000034  F6 E1                    mul     cl
00000036  D4 0A                    aam                          ; AX=0201h
                              ;-------
00000038  66| B8 0308              mov     ax,0308h
0000003C  B1 03                    mov     cl,3
0000003E  D5 0A                    aad
00000040  F6 F1                    div     cl
00000042  D4 0A                    aam                          ; AX=0102h

                              end
```

〔リスト4〕gasのDAA，DAS，AAA，AAS，AAM，AAD命令の記述例

```
1                       .text                                  ← 左の10進演算を行った場合の結果を示す
2   0000 B012                   movb    $0x12,%al
3   0002 0434                   addb    $0x34,%al
4   0004 27                     daa                             # CF=0, AL=46h
5                       #
6   0005 B098                   movb    $0x98,%al
7   0007 0459                   addb    $0x59,%al
8   0009 27                     daa                             # CF=1, AL=57h
9                       #-------
10  000a B034                   movb    $0x34,%al
11  000c 2C12                   subb    $0x12,%al
12  000e 2F                     das                             # CF=0, AL=22h
13                      #
14  000f B012                   movb    $0x12,%al
15  0011 2C38                   subb    $0x38,%al
16  0013 2F                     das                             # CF=1, AL=74h=-26h (100-74=26)
17                      #-------
18  0014 B400                   movb    $0,%ah
19  0016 B003                   movb    $0x03,%al
20  0018 0406                   addb    $0x06,%al
21  001a 37                     aaa                             # CF=0, AX=0009h
22                      #
23  001b B400                   movb    $0,%ah
24  001d B008                   movb    $0x08,%al
25  001f 0406                   addb    $0x06,%al
26  0021 37                     aaa                             # CF=1, AX=0104h
27                      #-------                                    減算の結果，
28  0022 B400                   movb    $0,%ah                      負になる場合の値
29  0024 B007                   movb    $0x07,%al
30  0026 2C05                   subb    $0x05,%al
31  0028 3F                     aas                             # CF=0, AX=0002h
32                      #
33  0029 B400                   movb    $0,%ah
34  002b B005                   movb    $0x05,%al
35  002d 2C07                   subb    $0x07,%al
36  002f 3F                     aas                             # CF=1, AX=FF08h=-02h (10-8=2)
37                      #-------
38  0030 B003                   movb    $0x03,%al
39  0032 B107                   movb    $0x07,%cl
40  0034 F6E1                   mulb    %cl
41  0036 D40A                   aam                             # AX=0201h
42                      #-------
43  0038 66B80803               movw    $0x0308,%ax
44  003c B103                   movb    $3,%cl
45  003e D50A                   aad
46  0040 F6F1                   divb    %cl
47  0042 D40A                   aam                             # AX=0102h
```

号付き整数として，正しく指定レジスタに格納できない(オーバフローした)場合，フラグの設定ではなく，ベクタ番号0の例外発生となります．例外が発生すると，割り込みベクタ番号0の例外処理ルーチンにジャンプすることになります．そのため，この例外を発生させたくない場合は，DIV命令と同様にIDIV命令の場合も，実行する前に被除数および除数を調べておく必要があります．

IDIV命令では，すべてのステータスフラグ(OF，SF，ZF，AF，PF，CF)の設定は未定義となっています．そのため，IDIV命令実行後のステータスフラグは使用できません．

10進算術命令

10進算術命令は，BCD(2進化10進数)で表される1桁あるいは2桁の整数値の四則演算を行うものです．演算に使用される値は符号なしですが，減算にかぎり演算結果として負数になる場合があります．

BCDの四則演算は，演算自体はこれまで述べた2進算術命令のADD，ADC，SUB，SBB，MUL，DIVを使用します．BCDの四則演算は，その2進算術命令の演算結果や演算前の値を補正することでBCDの四則演算を行います．つまり，10進算術命令は，この10進補正を行う命令ということになります．

10進算術命令には，パックBCDの加減算のためのDAA命令とDAS命令の2命令とアンパックBCDの加減乗除のためのAAA，AAS，AAM，AADの4命令，合計6個の命令があります(**表3**)．

この6個の10進算術命令は，使用するレジスタを固定しているため，すべて10進算術命令はオペランドをもちません．**リスト3**はMASMでの使用例，**リスト4**はgasでの使用例を示したものです．

● DAA命令とDAS命令

2桁のパックBCDの加減算を行う命令です．この場合，値は2桁のパックBCDなので長さは1バイトとなります．

DAA命令とDAS命令は，事前に2進算術命令の加減算を行っておく必要があります．また，DAA命令とDAS命令は，2進算術命令の加減算の結果，セットされるステータスフラグCFと

AF も 10 進補正に使用するため，2 進算術命令の加減算命令の実行後，すぐに DAA 命令や DSA 命令を実行する必要があります．

(1) DAA 命令

DAA 命令による 2 桁のパック BCD の加算では，事前に ADD，ADC 命令で加算し，その加算結果をレジスタ AL に格納しておきます．その上でこの DDA 命令で 10 進補正することで，レジスタ AL に 2 桁のパック BCD の加算結果を得ることができます．

DAA 命令実行後，10 進数としての桁上がりが CF(上 4 ビットからの) および AF(下 4 ビットからの) に設定されます．SF，ZF，PF は補正結果にしたがい設定されますが，OF は未定義となります．

2 桁より多いパック BCD を加算する場合は，必要なバイト数分，ADD，ADC 命令の加算と DDA 命令による 10 進補正を繰り返すことで行えます(**図 7**)．この場合，最上位桁の加算で発生した CF=1 はオーバフローを表すことになります．

(2) DAS 命令

DAS 命令による 2 桁のパック BCD の減算では，事前に SUB，SBB 命令で減算し，その減算結果をレジスタ AL に格納しておきます．その上でこの DSA 命令で 10 進補正することで，レジスタ AL に 2 桁のパック BCD の減算結果を得ることができます．

〔図 7〕DAA 命令による多数桁パック BCD 加算の例

DAS 命令実行後，10 進数としての桁借りが CF(上 4 ビット) および AF(下 4 ビット) に設定されます．SF，ZF，PF は補正結果にしたがい設定されますが，OF は未定義となります．10 進補正の結果 CF=1 だった場合，パック BCD は負数となります．この場合は，負数は 10 の補数により表されます(**図 8**)．

2 桁より多いパック BCD を減算する場合は，必要なバイト数分，SUB，SBB 命令の減算と DAS 命令による 10 進補正を繰り返すことで行えます(**図 9**)．この場合，最上位桁の減算で発生した CF=1 は 10 の補数による負数を表すことになります．

● AAA 命令と AAS 命令

1 桁同上のアンパック BCD の加減算に使用します．この場合，値は 1 桁のアンパック BCD なので長さは 1 バイトとなります．

AAA 命令と AAS 命令は，事前に 2 進算術命令の加減算を行っておく必要があります．また，AAA 命令と AAS 命令は，2 進算術命令の加減算の結果，セットされるステータスフラグ AF も 10 進補正に使用するため，2 進算術命令の加減算命令の実行後，すぐに AAA 命令や AAS 命令を実行する必要があります．

(1) AAA 命令

1 桁のアンパック BCD の加算では，まず ADD，ADC 命令で加算し，その結果をレジスタ AL に格納しておきます．その上でこの AAA 命令で 10 進補正することで，レジスタ AL に 1 桁のアンパック BCD の加算結果を得ることができます〔**図 10**(**a**)〕．このとき，レジスタ AL の上位 4 ビットはゼロになっています．

〔図 8〕10 の補数

(**a**) BCD1桁の場合の「10の補数」による表現　(**b**) BCD n 桁の場合の「10の補数」による表現

AAA命令実行後，10進数としての桁上がりが発生した場合は，レジスタAHは＋1され，CF=AF=1に設定されます．桁上がりがなかった場合は，レジスタAHは変化せず，CF=AF=0に設定されます．

CFとAF以外のフラグ(OF，SF，ZF，PF)は未定義となります．

(2) AAS命令

1桁のアンパックBCDの減算では，まずSUB，SBB命令で減算し，その結果をレジスタALに格納しておきます．その上でこのAAS命令で10進補正することで，レジスタALに1桁のアンパックBCDの減算結果を得ることができます〔**図10(b)**〕．このとき，レジスタALの上位4ビットはゼロになっています．

AAS命令実行後，10進数としての桁借りが発生した場合は，レジスタAHは－1され，CF=AF=1に設定されます．桁借りがなかった場合は，レジスタAHは変化せず，CF=AF=0に設定されます．

CFとAF以外のフラグ(OF，SF，ZF，PF)は未定義となります．

● AAM命令

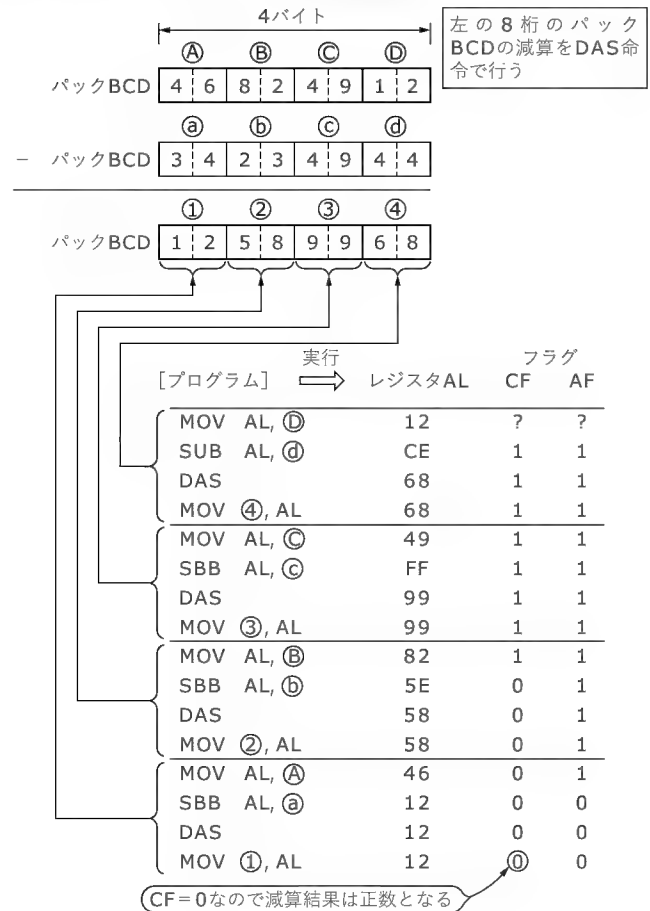
1桁同士のアンパックBCDの乗算に使用します．この場合，値は1桁のアンパックBCDなので長さは1バイトとなります．また，乗算される二つの1桁のアンパックBCDは，ともに上位4ビットをゼロにしておきます．

1桁のアンパックBCDの乗算では，まずMUL命令でバイト乗算し，その結果をレジスタAXに格納しておきます．その上でこのAAM命令で10進補正することで，レジスタAXに2桁のアンパックBCDの乗算結果を得ることができます〔**図11(a)**〕．このときレジスタAHが上の桁，レジスタALが下の桁となります．

AAM命令実行によりSF，ZF，PFのフラグは結果にしたがって設定されます．残りのOF，AF，CFは未定義となります．

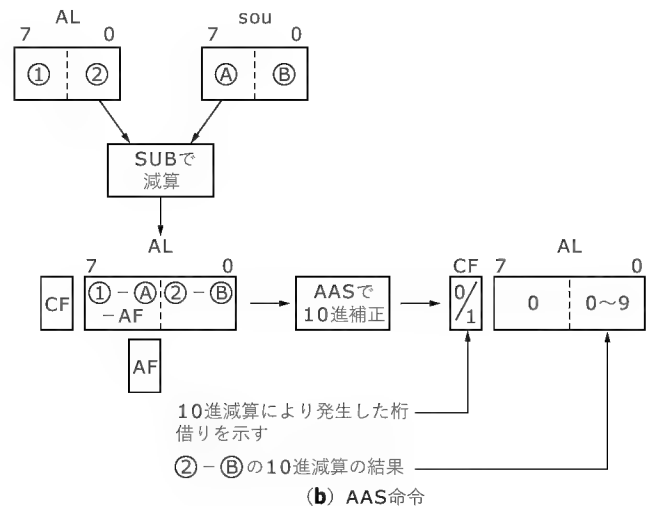
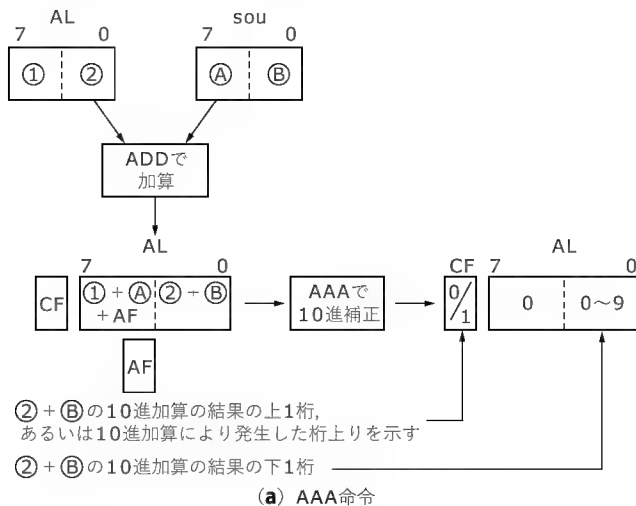
● AAD命令

2桁÷1桁のアンパックBCDの除算に使用します．この場合，

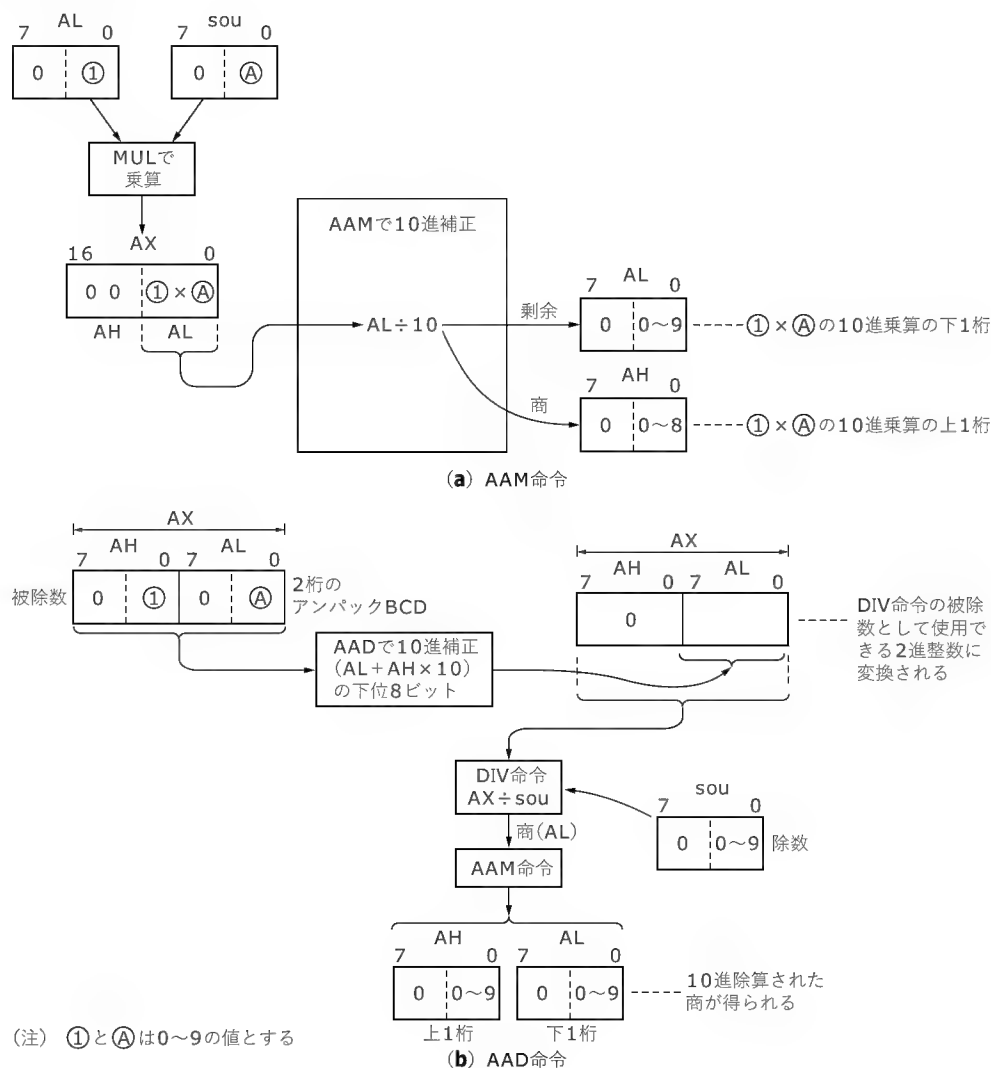
〔図9〕DASによる多数桁パックBCD減算の例

被除数は2桁のアンパックBCDなので長さは2バイトとなり，除数は1桁のアンパックBCDなので長さは1バイトとなります．また，被除数，除数とも1桁分のアンパックBCDは，上位4ビットをゼロにしておきます．

〔図10〕AAA命令，AAS命令の動作

(a) AAA命令

(b) AAS命令

(注) ②とⒷは0～9の値とする

〔図11〕AAM命令，AAD命令の動作

（a）AAM命令

（b）AAD命令

AAD命令は，DIV命令を実行する前に使用します．AAD命令を実行する場合，被除数となる2桁のアンパックBCDは，レジスタAHに上の桁，レジスタALに下の桁を設定しておきます．除数は上位4ビットがゼロになっていることが前提なので，そのままDIV命令の除数として使用できます．その上で，AAD命令を実行するとレジスタALに，2桁のアンパックBCDを2進整数に変換した値が設定され，レジスタAHはゼロになります．

この後，16ビット÷8ビットのDIV命令で2進除算を行い，最後にAAM命令を実行することで，レジスタAXには2桁のアンパックBCDの除算結果(商)を得ることができます〔**図11(b)**〕．このときレジスタAHが上の桁，レジスタALが下の桁となります．

*　　　　　*

次回は論理演算とシフト，ローテート命令について説明する予定です．

おおぬき・ひろゆき　大貫ソフトウェア設計事務所

TMS320C6711搭載DSPスタータキットと
PCM3003搭載オプションボードを使った

ステレオオーディオDSPプログラミング入門（基礎編）

三上直樹

はじめに

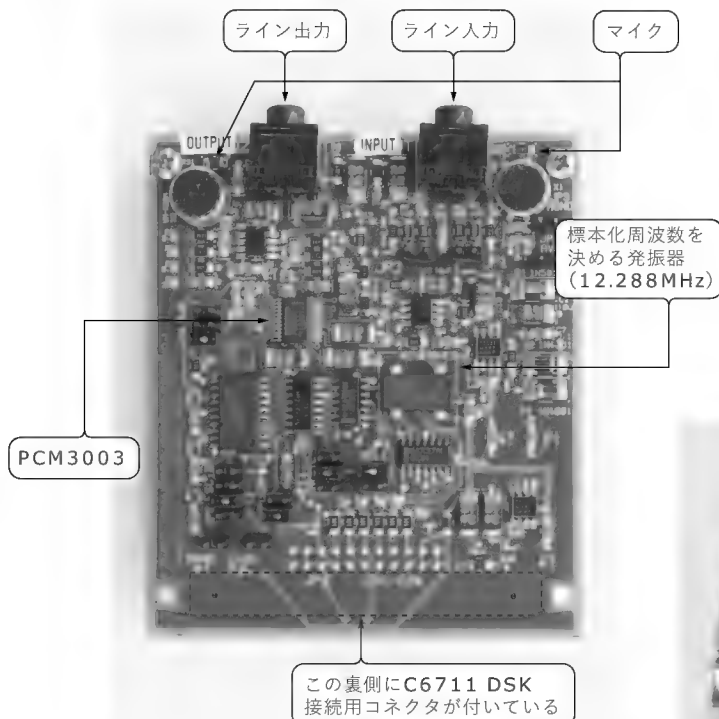
テキサスインスツルメンツ社のTMS320C6711は，アドバンストVLIWアーキテクチャを採用した浮動小数点演算タイプのDSPです．このTMS320C6711を搭載したC6711 DSKは，ディジタル信号処理をリアルタイムで手軽に実行するのに便利なキットです[1]．

筆者の勤務する大学でも，このDSKを"信号処理工学実習"で利用しています．これを使うと，実際に処理結果を音として耳で聞くことや，オシロスコープを使って波形を見ることができ，学生にも好評です．

C6711 DSKには，アナログ信号入出力のためのCODECが搭載されていますが，標本化周波数が8kHzに固定であり，また入出力がともに1チャネルしかないので，用途が限られていました．しかし最近，このDSKで使うことができるPCM3003（バーブラウン[注1]）が搭載されたオーディオ用のドータカード[注2]を使用する機会を得ました．これにはA-D変換器とD-A変換器が2チャネル内蔵され，さまざまな用途に使うことができます．そこで，そのドータカードの解説を2回にわたって行います．今回は，主としてその基本的なプログラミングについて解説を行います．

なお，TMS320C6711とそれを搭載したDSPスタータキット"C6711 DSK"については**コラム**を参照してください．

〔写真1〕オーディオ用ドータカードの外観

(a) TMDX326040A

(b) C6711 DSKとスタック接続したようす

1. オーディオ用ドータカード（TMDX326040A）のハードウェア

このドータカードにはバーブラウンのPCM3003[2]が搭載され，C31 DSKとC6711 DSKに対応しています．**写真1(a)**にこのドータカードを示します．アナログ信号の入出力は，ステレオピンジャックにより行います．そのほかに，2個のマイクが搭載されています．このカードのおもな仕様を**表1**に示します．

注1：バーブラウン社は，現在テキサスインスツルメンツ社に統合されている．
注2：製品名はTMDX326040Aで，その概要についてはテキサスインスツルメンツ社のホームページ（http://www.ti.com/）から探すことができる．それによると米国での価格は$50となっている．日本テキサスインスツルメンツ社の代理店を通して購入することができる．なお，低価格化を図るため，このドータカードにはマニュアル類が付いてこないが，ftp://ftp.ti.com/pub/cs/c6000/DSK/AudioDC/よりマニュアル類をダウンロードできる．

〔表1〕オーディオ用ドータカードTMDX326040Aの仕様

搭載のCODEC	バーブラウンPCM3003
入出力チャネル数	それぞれ2チャネル
ライン入力/出力端子	オーディオ用ステレオミニジャック，最大入力：1Vp-p，AC結合
マイク	2個
標本化周波数（内蔵クロック使用の場合）注	24kHz，48kHzの選択
ビット数	16ビット，20ビットの選択
対応可能DSK	C31DSK，C6711DSK
C6711DSKとの接続	DSPのMcBSP1を利用

注：外部クロックはDSPのタイマより供給可能

注3：ダウンロードしたマニュアル[3]に示されるジャンパピンの1番ピンの位置は，実際のものとずれている．ボードの裏から見たときに，スルーホールのパターンが四角形になっているところが1番ピンに対応する．

PCM3003はA-D変換器とD-A変換器をそれぞれ2チャネル持っているため，アナログ信号を同時に標本化することができます．このA-D変換器およびD-A変換器は20ビットで64倍オーバサンプリングの⊿Σ方式を採用しています．入出力データは16ビット，20ビットの切り替えができます．

このドータカードには，いくつかのジャンパピンが付いています．以降で使用するための，ジャンパピンの短絡用ソケット設定のようすを**図1**(p.160)に示します[注3]．この設定で，標本化周波数は48kHz，データのビット数は16ビットに設定されます．各ジャンパピンの機能と設定状態について**表2**に示します(網掛けの部分はデフォルト設定とは異なるところ)．

JP1，JP2には，デフォルトで短絡用ソケットが設定されていますが，この設定はカードに搭載されているマイクが接続され

TMS320C6711と DSPスタータキット“C6711 DSK”について

● TMS320C6711のハードウェアの概要

現在，アドバンストVLIWアークテクチャを採用するTMS320C6000シリーズのDSPには，三つのグループのDSPが発表されています．固定小数点演算方式のTMS320C62xシリーズ，浮動小数点演算方式のTMS320C67xシリーズ，そしてTMS320C62xシリーズをさらに高性能にしたTMS320C64xシリーズです．

TMS320C6711は，TMS320C67xシリーズのDSPの一つです．

表Aに，TMS320C6711のおもな性能を示します．この表からわかるように，このDSPはクロック周波数150MHzで900MFLOPSという非常に高い浮動小数点演算性能をもっています．また，1サイクルあたり8命令を並列に実行することができ，1200MIPSの処理能力をもっています．

図Aに，TMS320C6711のブロック図を示します．全体は，CPUコア，内部メモリ，ペリフェラルに大きく分けることができます．CPUコアはデータ処理を行う際の中心部です．CPUコアは，命令フェッチなどの命令の操作を行う部分，データ処理の中心となるデータパス，および制御レジスタなどから構成されています．

〔表A〕TMS320C6711のおもな性能

最小マシン・サイクル	6.67ns（クロック周波数：150MHz）
最小処理能力	1200MIPS*，8命令/サイクル
最大演算能力	900MFLOPS**，6演算/サイクル
データ形式	8/16/32ビット，固定小数点 32/64ビット，浮動小数点
命令形式	32ビット×8命令
アドレス空間	4Gバイト（アドレスはバイト単位）
内蔵メモリ	命令用1次キャッシュ　4Kバイト データ用1次キャッシュ　4Kバイト 2次キャッシュ/RAM　64Kバイト
汎用レジスタ	32ビット×16個×2組
機能ユニット	8
製造プロセス技術	0.18μm，CMOS
電源電圧	3.3V（入出力），1.8V（内部）
パッケージ	256ピンBGA（ball grid array）

*：Million instructions per second
**：Mega floating-point instructions per second

内部メモリは，2レベルのキャッシュメモリになっています．1次キャッシュはデータ用(L1D)とプログラム用(L1P)に分かれており，それぞれ4Kバイトの大きさです．2次キャッシュ(L2)はデータ/プログラム共用で，64Kバイトの大きさです．2次キャッシュの領域は，リセット時には通常のメモリとして設定されています．2次キャッシュをキャッシュとして利用するためには，モードの切り替えが必要になります．

2次キャッシュとペリフェラルとのデータのやり取りはエンハンスドDMA(EDMA)コントローラを通して行います．

外部メモリとのデータの授受は外部メモリインターフェース(EMIF)を通して行います．なお，パラレルのI/Oポートはとくに設けられておらず，この外部メモリインターフェースを使います．いわゆるメモリマップドI/Oになっています．

マルチチャンネルバッファードシリアルポート(McBSP)は同期式のシリアルポートで2組あり，それぞれ128チャネルまで扱うことが可能です．ホストポートインターフェース(HPI)は16ビット幅で，ホストコンピュータとのデータのやり取りに使われます．タイマは32ビット幅で2組あり，外部イベントや内部クロック×1/4を基準としてカウントを行います．

● C6711 DSKのハードウェア

図BにC6711 DSKボードの主要部のブロック図を示します．C6711 DSKボードに搭載されているTMS320C6711には150MHzのクロックが供給され，最大で1200MIPS/900MFLOPSの能力をもちます．その他，100MHzのクロックが供給されていますが，これは外部メモリのインターフェースのためのものです．電源も2系統になっており，コア部へは1.8V，I/O部へは3.3Vが供給されます．これらは，ボードに供給されている5Vから，オンボードのスイッチングレギュレータによって作られています．

ボードに搭載されているメモリは，16ビット×4MバイトのSDRAMが二つと，8ビット×128Kバイトのフラッシュメモリが一つです．その他，メモリおよびペリフェラルを拡張するためのインターフェース回路と，ドータボード用のコネクタがついています．これにより，ユーザーがメモリやペリフェラルを拡張することも可能です．

このボードとプログラム開発用のPCとの接続は，パラレルポート

た状態になります．したがって，マイクを使わない場合には，マイクから不要な信号が入らないように，短絡用ソケットをはずしておきます．

一方，JP4には，デフォルトで短絡用ソケットが設定されていませんが，ここには**図1**に示すように短絡用ソケットを設定します．この設定により，PCM3003のディエンファシスの機能がキャンセルされ，通過域の振幅特性がほぼ平坦になります注4．

JP5は標本化周波数を決めるクロックとして，内蔵のクロックを使うか，DSPから供給するかの切り替えを行います．**図1**の設定で，内蔵クロックを使うように設定されます．DSPからクロックを供給する場合は，TMS320C6711のタイマ0を使います．

注4：ダウンロードしたマニュアル[3]のJP4の説明には誤りがある．

〔表2〕TMDX326040Aのジャンパピンの設定

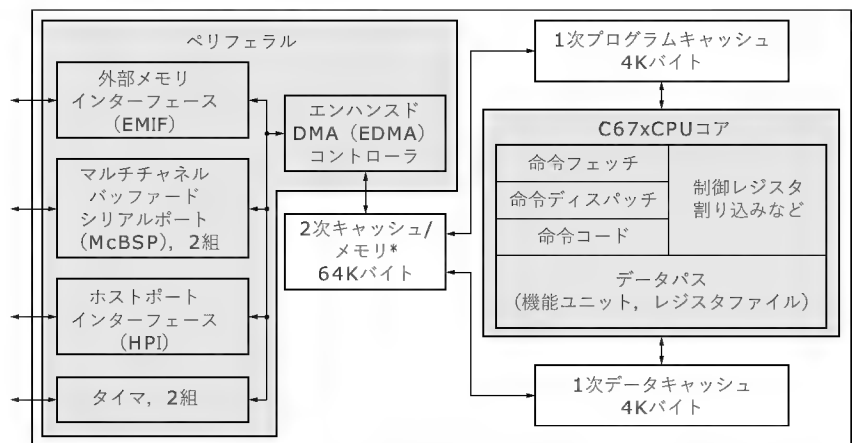
ジャンパ	機　能	筆者の設定	デフォルトの設定
JP1	右マイクの接続/切り離しの切替	なし	1-2
JP2	左マイクの接続/切り離しの切替	なし	1-2
JP3	C31 DSK用のコネクタとして使用		
JP4	PCM3003のディエンファシス特性の設定	5-6	なし
JP5	標本化クロックの基準の切替	3-4	3-4
JP6	フレーム同期のタイミングの切替	3-4	3-4
JP7	ディジタルグランド	なし	なし
JP8	ディジタル3.3V	なし	なし
JP9	アナロググランド	なし	なし
JP10	アナログ3.3V	なし	なし
JP11	ビット長とビットレートの設定	3-4	5-6注
JP12	ビット長とビットレートの設定	1-2	3-4注

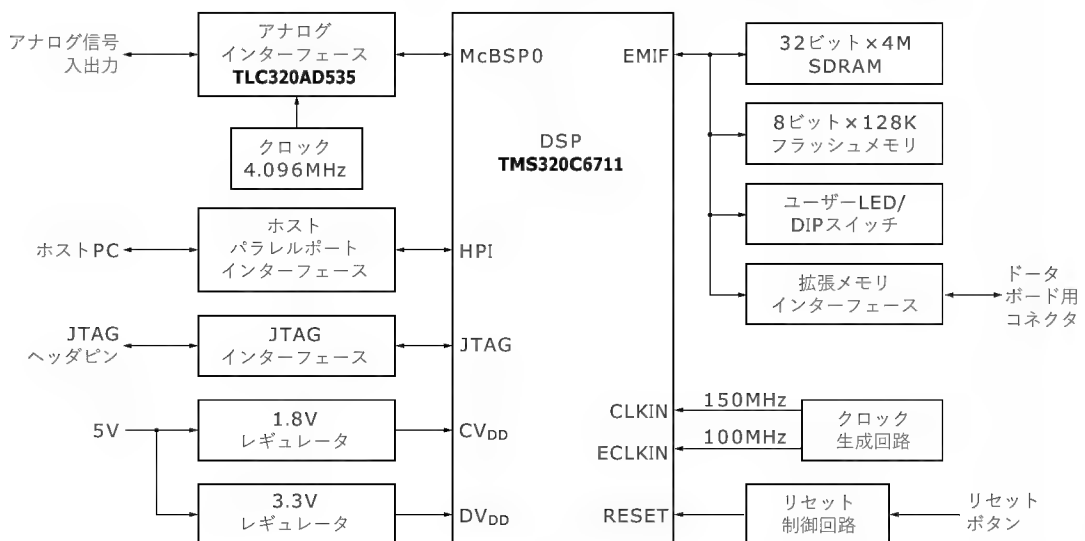
注：この設定の場合は，標本化周波数が24kHzに設定される．

(IEEE1284準拠，25ピンD-Subコネクタ)を通して行われます．

アナログ信号の入出力のためには，A-D変換器とD-A変換器を内蔵するTLC320AD535が搭載されています．このTLC320AD535には，A-D変換器の前段のアンチエリアシングフィルタ，D-A変換器の後段のスムージングフィルタも内蔵されています．標本化周波数は8kHzに固定されており変更はできません．

その他，ボードをリセットするためのプッシュスイッチおよびユーザーが自由に使うことのできるディップスイッチとLEDがそれぞれ搭載されています．JTAGヘッダピンは，TI社XDS510エミュレータなどを接続してプログラム開発するときに利用するためのものです．

〔図A〕TMS320C6711のブロック図

*：2次キャッシュ/メモリはキャッシュとしてだけではなく，通常のプログラムメモリまたはデータメモリとしても使用可能

〔図B〕C6711 DSKのブロック図

・McBSP：multichannel buffered serial port
・HPI　：host port interface
・EMIF　：external memory interface

JP11，12は**図1**の設定で，標本化周波数を48kHzに，データのビット数を1チャネルあたり16ビットに設定します．

C6711 DSKと組み合わせる場合は，C6711のシリアルポート1(McBSP1)を通してDSPとのデータの受け渡しを行います．**写真1(b)**には，このカードをC6711 DSKにセットしたようすを示します．

なお，搭載されているPCM3003のアナログ入力部には，エリアシング防止用低域通過フィルタと直流分をカットするための高域通過フィルタが内蔵されています．また，PCM3003のアナログ出力部には，信号の平滑化のための低域通過フィルタが内蔵されています．これらのフィルタを無効にすることはできません．さらに，カードと外部をつなぐアナログ入力および出力はAC結合になっているため，直流を扱うことはできません．また，アナログ出力部には遮断周波数が27kHzの2次の低域通過フィルタが設けられていますが，これはノイズ防止用のものと思われます．

2. オーディオ用ドータカードを使うための基本的なプログラムの開発(ポーリング方式の場合)

オーディオ用ドータカードを使うための基本的なプログラムを作成します．一つはPCM3003とのデータのやり取りをポーリング方式で行うもので，もう一つは割り込み方式で行うものです．

〔**図1**〕**TMDX326040Aのジャンパピンの位置**

C6711 DSKに付属するCode Composer Studio注5 (以下CCS)のコンパイラはC言語およびC++言語に対応しています注6．そこで，本稿ではリセットベクタを除くソースプログラムをC++言語で記述することにします[4]．

2.1 ポーリング方式で使用するクラス

ポーリング(polling)方式とは，周辺機器(ここではMcBSP1)の状態をCPUが監視をして入出力が可能かどうかを調べ，可能なときにデータの転送を行う方式です．

リスト1は，ポーリング方式でアナログ信号の入出力を行うための`PCM3003`という名前のクラス(class)です．このクラスはアナログ信号の入出力のためのメンバ関数のほかに，DSPの初期設定に使うメンバ関数などが含まれます．これらに対して，`PCM3003_Polling.cpp`というファイル名を付けています．このファイルは以降でも共通に使います．

このクラスのpublic部では，コンストラクタの`PCM3003()`，アナログ信号を入力するためのメンバ関数`ReadRdy()`，アナログ信号を出力するためのメンバ関数`Write()`が宣言されています．アナログ信号の入出力のためのメンバ関数はC++言語の機能の一つである，関数の多重定義(オーバロード：overload)の機能を使っています．

以下では，コンストラクタおよびアナログ信号入出力用のメンバ関数について説明します．

● `PCM3003()`

コンストラクタでは，DSPの初期設定と，DSPとドータカードとのデータのやり取りに使うMcBSP1の初期設定を行っています．

▶ DSPの初期設定

DSPの初期設定としては，すべてのマスカブル(maskable)割り込みの禁止，すべての割り込みの禁止，DSP内蔵の2次キャッシュの設定を行います．

2次キャッシュの設定は，`private`メンバ関数`CacheSet()`で行われます．この関数では，キャッシュとして使う領域のサイズも設定され，このサイズはコンストラクタの引き数で決定されます．コンストラクタの引き数は`bank_num`型で与えます．`bank_num`型の数値は，キーワード`enum`を使って定義した列挙型です．

コンストラクタは，デフォルトの引き数が定義されています．したがって，コンストラクタに引き数を与えない場合は，2次キャッシュのサイズは可能な最大の大きさ(64Kバイト)に設定されます．

注5：バージョン2.1に基づいて説明する．
注6：ここではバージョン4.2のC/C++コンパイラに基づいて説明する．拡張子が`.c`であればCのソースプログラム，`.cpp`であればC++のソースプログラムとみなされる．

〔リスト1〕**DSPの初期化およびポーリング方法でアナログ信号の入出力を行うプログラム**(PCM3003_Polling.cpp)

```
//---------------------------------------------
//  class for using PCM3003 daughter card
//         by MIKAMI, Naoki  2002/12/12
//      In little-endian mode:
//          ch0 or ch[0] : right channel
//          ch1 or ch[1] : left channel
//---------------------------------------------
#ifndef MK_PCM3003_Polling

#include <c6x.h>
#include <c6x11dsk.h>

typedef volatile unsigned int u_v_int;
typedef volatile short v_short;

enum banks_num { cach_bk0, cach_bk1, cach_bk2, cach_bk3,
                                                  cach_bk4=7 };

class PCM3003
{
private:
    void CacheSet(banks_num banks);
    void McBSP1_init();
    inline unsigned int McBSP1ReadRdy();
    inline void McBSP1Write(unsigned int data);
protected:
    union { unsigned int buf; short chn[2]; } xn, yn;
public:
    PCM3003(banks_num banks = cach_bk4);
    inline void ReadRdy(short &ch0, short &ch1);
    inline void ReadRdy(float &ch0, float &ch1);
    inline void ReadRdy(short ch[]);
    inline void ReadRdy(float ch[]);
    inline void Write(short ch0, short ch1);
    inline void Write(float ch0, float ch1);
    inline void Write(short ch[]);
    inline void Write(float ch[]);
};

// Constructor: Initialize some control registers and McBSP1
PCM3003::PCM3003(banks_num banks)
{
    CSR = 0x100;         // Disable all maskable interrupts
    IER =0;              // Disable all interrupts
    CacheSet(banks);     // 2nd cache set and enable
    McBSP1_init();       // Initialization of McBSP1
}

// Receive data from PCM3003 (data type: short)
inline void PCM3003::ReadRdy(short &ch0, short &ch1)
{
    xn.buf = McBSP1ReadRdy();
    ch0 = xn.chn[0];
    ch1 = xn.chn[1];
}

// Receive normalized data from PCM3003 (data type: float)
//      -1.0 <= data < 1.0
inline void PCM3003::ReadRdy(float &ch0, float &ch1)
{
    xn.buf = McBSP1ReadRdy();
    ch0 = (float)xn.chn[0]*3.051758e-5f;
    ch1 = (float)xn.chn[1]*3.051758e-5f;
}

// Receive data from PCM3003 (data type: short[])
inline void PCM3003::ReadRdy(short ch[])
{
    xn.buf = McBSP1ReadRdy();
    for (int k=0; k<2; k++) ch[k] = xn.chn[k];
}

// Receive normalized data from PCM3003 (data type: float[])
//      -1.0 <= data| < 1.0
inline void PCM3003::ReadRdy(float ch[])
{
    xn.buf = McBSP1ReadRdy();
    for (int k=0; k<2; k++) ch[k] =
                                 (float)xn.chn[k]* 3.051758e-5f;
}

// Transmit data to PCM3003 (data type: short)
inline void PCM3003::Write(short ch0, short ch1)
{
    yn.chn[0] = ch0;
    yn.chn[1] = ch1;
    McBSP1Write(yn.buf);
}

// Transmit normalized data to PCM3003 (data type: float)
inline void PCM3003::Write(float ch0, float ch1)
{
    yn.chn[0] = (short)(ch0*32768.0f);
    yn.chn[1] = (short)(ch1*32768.0f);
    McBSP1Write(yn.buf);
}

// Transmit data to PCM3003 (data type: short[])
inline void PCM3003::Write(short ch[])
{
    for (int k=0; k<2; k++) yn.chn[k] = ch[k];
    McBSP1Write(yn.buf);
}

// Transmit normalized data to PCM3003 (data type: float[])
inline void PCM3003::Write(float ch[])
{
    for (int k=0; k<2; k++) yn.chn[k] = (short)(ch[k]*32768.0f);
    McBSP1Write(yn.buf);
}

// Set cache size and turn on caching CE0
//      banks must be cach_bk0, cach_bk1, cach_bk2,
//                                   cach_bk3, or  cach_bk4
void PCM3003::CacheSet(banks_num banks)
{
    *(u_v_int *)L2CFG = banks;   // set number of banks for cache
    *(u_v_int *)MAR0 = 1;
                      // Turn on cashing 0x80000000 - 0x80FFFFFF
}

// McBSP1 Initialization for PCM3003
void PCM3003::McBSP1_init()
{
    *(u_v_int *)McBSP1_SPCR = 0;            // reset serial port
// set pin control register  FSX, FSR : active low
    *(u_v_int *)McBSP1_PCR  = 0x000C;
// 0 bit data delay, 1 word/frame, Receive element length
                                                   = 32 bits
    *(u_v_int *)McBSP1_RCR  = 0x00A0;
// 0 bit data delay, 1 word/frame, Transmit element length
                                                   = 32 bits
    *(u_v_int *)McBSP1_XCR  = 0x00A0;
    *(u_v_int *)McBSP1_DXR = 0;             // write first word
    *(u_v_int *)McBSP1_SPCR = 0x10001;     // serial port enable
    u_v_int dummy = *(u_v_int *)McBSP1_DRR; // dummy read
 }

// Receive data from PCM3003 using McBSP1(with ready flag check)
inline unsigned int PCM3003::McBSP1ReadRdy()
{
    while ( (*(u_v_int *)McBSP1_SPCR & 0x2) != 0x2);
    return ( *(u_v_int *)McBSP1_DRR );
}

// Transmit data to PCM3003 using McBSP1
//      upper 16 bits: left channel
//      lower 16 bits: right channel
inline void PCM3003::McBSP1Write(unsigned int data)
{
    *(u_v_int *)McBSP1_DXR = data;
}

#define MK_PCM3003_Polling
#endif
```

▶ McBSP1の初期設定[5)]

McBSP1の初期設定は，privateメンバ関数McBSP1()で行われます．

ピンコントロールレジスタ(PCR)の設定では，受信フレーム同期極性(FSRP，第3ビット目)と送信フレーム同期極性(FSXP，第2ビット目)の各ビットを1に設定し，フレーム同期パルスがアクティブローになるように設定します[注7]．この設定は，次の文で行っています[注8]．

```
*(u_v_int *)McBSP1_PCR=0x000C;
```

次に，受信/送信コントロールレジスタの設定ですが，ここではドータカードに搭載されているPCM3003とのデータのやり取りを，32ビット単位で行うように設定しています．つまり，1チャネル当たり16ビットのデータを2チャネル分一括して転送するように設定されます．このような設定を行うため，受信コントロールレジスタ(PCR)と送信部コントロールレジスタ(XCR)へ次の二つの文で書き込みを行っています．

```
受信部の設定  *(u_v_int *)McBSP1_RCR=0x00A0;
送信部の設定  *(u_v_int *)McBSP1_XCR=0x00A0;
```

この設定の意味は，**図2**に示します．

● ReadRdy( )

PCM3003の2チャネル分のA-D変換されたデータを読み込むためのメンバ関数です．このメンバ関数は，PCM3003から16ビットで2チャネル分のデータがMcBSP1に転送完了したことを確認して，そのデータを取り込むようになっています．

転送完了の確認は，シリアルポートコントロールレジスタ(SPCR)の第1ビット目が1になっていることで行います．これらの処理のために，privateメンバ関数McBSP1ReadRdy()を使っています．このメンバ関数は2チャネル分のデータを一括して読み込み，これをunsigned int型の戻り値とします．

McBSP1ReadRdy()から受け取るunsigned int型データには2チャネル分のshort型のデータが含まれているため，これを分離する必要があります．そこで，一つのunsigned int型変数に対して，その上位と下位の16ビットをそれぞれshort型変数が対応するように，protected部ではキーワードunionを使って共有型の変数xn，ynを宣言しています[注9]．

メンバ関数ReadRdy()は多重定義されており，short型とfloat型のそれぞれの単純変数と配列が使えるようになっています．その宣言は次のようになっています．

```
inline void ReadRdy(short &ch0, short &ch1);
inline void ReadRdy(float &ch0, float &ch1);
inline void ReadRdy(short ch[]);
inline void ReadRdy(float ch[]);
```

なお，ch0またはch[0]がステレオの右チャネルに，ch1またはch[1]がステレオの左チャネルに対応しています．

このメンバ関数は，引き数がshort型かfloat型かにより，受け取るデータが異なります．引き数がshort型の場合は，PCM3003でA-D変換されたデータそのままです．一方，float型の場合は，$-1 <=$ データ < 1 に正規化するため，受け取ったデータに 2^{-15} を乗算したものになっています．

● Write( )

PCM3003へ2チャネル分のデータを書き込むためのメンバ関数です．2チャネル分のデータを一括してMcBSP1に転送するためには，privateメンバ関数McBSP1Write()を使っています．

メンバ関数Write()も多重定義されており，その宣言は次のようになっています．

```
inline void Write(short ch0, short ch1);
inline void Write(float ch0, float ch1);
inline void Write(short ch[]);
inline void Write(float ch[]);
```

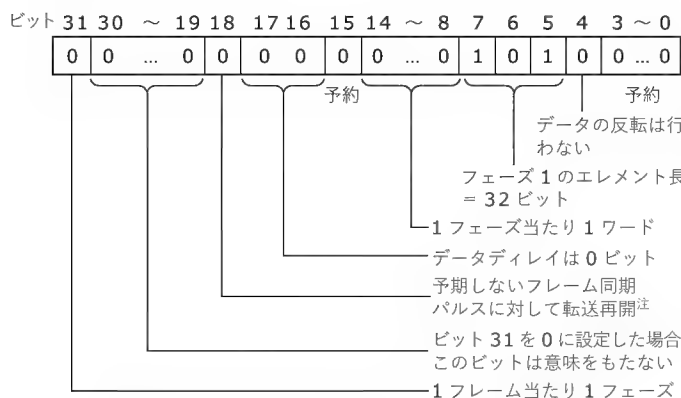
引き数はReadRdy()の場合と同じ関係になっています．ただし，float型の場合は，2^{15} を乗算したものをPCM3003のD-A変換器に送り出します．

〔図2〕McBSPの受送信コントロールレジスタ(RCR/XCR)の設定

2.2 ポーリング方式で使用するクラスの使用例①

DSP上で実行可能なコードの入った実行可能ファイル(拡張子：.out)をビルドするためには，信号処理をC/C++言語で記述したプログラムのほかに，リセットベクタや割り込みベクタについてアセンブリ言語により記述したファイル(拡張子：.asm)と，リンカコマンドファイル(拡張子：.cmd)が必要になります．

(1) リセットベクタ用ファイル

リスト2に，リセットベクタ用のソースプログラム(vects_

注7：アクティブハイに設定すると，二つのチャネルの時間関係が1サンプル分(標本化周期に等しい時間)だけ異なるようになる．これは，PCM3003のLRCIN端子の入力されるクロックが，DSPのFSR1，FSX1端子へはインバータ(74AHC04)を通して反転されて入力されていることによる．

注8：u_v_intは“typedef volatile unsigned int u_v_int;”というように宣言している．

注9：この変数は派生クラスでも使うため，宣言文をprotected部に置いている．

Reset.asm)を示します．通常は割り込みベクタ用の記述も必要ですが，ここでは割り込みを使わないので，リセットベクタに対応する部分のみが書かれています注10.

このプログラムでは.sectという疑似命令でvectorsという名前のセクションを定義しています．そのため，リセットベクタはvectorsというセクションに配置されることになります．セクションは論理的なもので，このセクションとDSP上の実際のアドレスとの対応関係は，次のリンカコマンドファイルで指定します．

以降で作成するプログラムでも，割り込みを使用する場合を除き，**リスト2**のファイルを使います．

(2) リンカコマンドファイル

リスト3に，リンカコマンドファイル(lnk_std.cmd)の内容を示します．**リスト2**ではvectorsというセクションを定義しましたが，これに対応するDSP上の実アドレスの先頭が，リンカコマンドファイルにより0x00000000に指定されます．また，プログラムのその他の部分は，すべて0x80000000から順に配置されることになります．TMS320C6711では，この領域は外部RAMに割り当てられています．その他，オプションとしてスタック領域とヒープ領域のサイズ，およびランタイムライブラリとしてrts6700.libを使うという指定を行っています．

なお，以降でも**リスト3**に示すリンカコマンドファイルをそのまま使います．

(3) C++言語によるソースプログラム

まず，2.1で説明したクラスを使って作成した，もっとも基本的なプログラム(Stereo_Through.cpp)を**リスト4**に示します．このプログラムは，PCM3003のA-D変換器から入力された信号を，何も処理を行わずにそのままPCM3003のD-A変換器に送り出すというものです．

最初に，“PCM3003_Polling.cpp”をインクルードする必要があります．クラスPCM3003によるオブジェクトの宣言では，引き数を省略しています．したがって，**リスト1**のコンストラクタに関する記述の部分からわかるように，この場合は2次キャッシュ領域が最大の64Kバイトに設定されます．

DSPの初期設定とMcBSP1の初期設定は，すべてクラスPCM3003のコンストラクタが行うため，main()関数では初期設定に関する処理を何も行いません．while文による無限ループとなっている部分では，アナログ信号の入力をメンバ関数ReadRdy()で行い，その信号に対して何も処理せずにそのまま，メンバ関数Write()で出力しています．

2.3 ポーリング方式で使用するクラスの使用例②

2.2ではもっとも基本となるプログラムを示しましたが，次にボーカルを含むCDの音楽から，ボーカルを取り除くというプログラムを示します．ここで行う処理の原理は非常に簡単です．多くの場合，ボーカルの成分は左右のチャネルで同じ位相になります．したがって，左右のチャネルから来た信号を同時に標本化を行い，その差を出力すればボーカルを取り除くことができます．

この方法では，当然ですがボーカル成分の位相が左右で違う場合には，ボーカルをうまく取り除くことはできません．したがって，CDの録音条件によっては，ボーカルを取り除けない場

〔リスト2〕リセットベクタ(vects_Reset.asm)

```
;******************************
;       Vector for Reset       *
;******************************
        .sect "vectors"
        .ref  _c_int00          ; C/C++ entry point

RESET:                          ; reset vector
        MVKL  _c_int00,B0       ; B0: start address of _c_int00
        MVKH  _c_int00,B0
        B     B0                ; branch to address pointed by B0
        NOP
        NOP
        NOP
        NOP
        NOP
```

〔リスト3〕リンカコマンドファイル(lnk_std.cmd)

```
-stack 0x400
-heap  0x400
-l rts6700.lib

MEMORY
{
    vects: org = 0x00000000,  len = 0x00000200
    SDRAM: org = 0x80000000,  len = 0x01000000
}

SECTIONS
{
    vectors  > vects
    .text    > SDRAM
    .bss     > SDRAM
    .cinit   > SDRAM
    .const   > SDRAMa
    .stack   > SDRAM
    .cio     > SDRAM
    .sysmem  > SDRAM
    .far     > SDRAM
}
```

〔リスト4〕ポーリング方式でA-D変換器からの入力信号をそのままD-A変換器に出力するプログラム(Stereo_Through.cpp)

```
//---------------------------------------------
//  Stereo signal through program using polling
//---------------------------------------------
#include "PCM3003_Polling.cpp"

int main()
{
    PCM3003 codec;
    short ch0, ch1;

    while (1)    // endless loop
    {
        codec.ReadRdy(ch0, ch1);
//--------------------------------------------
//  Put digital signal processing routine here
//--------------------------------------------
        codec.Write(ch0, ch1);
    }
}
```

注10：割り込みを使う場合には，割り込みに対応するベクタの部分を作成する必要がある．これについては3.2で説明する．

合も出てきます．また，左右のチャネルの伝達特性には多少の差があるので，ボーカルを完全には取り除くことができません．これらの理由から，ボーカルが小さく聞こえる場合もあります．

作成するプログラムでは，DSKボードのディップスイッチによりボーカル除去処理の有効/無効を設定できるようにします．また，ボーカル除去処理が有効なときは，DSKボードのLEDを点灯させるようにします．

作成したプログラム(`VocalCanceler.cpp`)を**リスト5**に示します．スイッチおよびLEDのアドレスに対応するシンボルは`IO_PORT`で，このアドレスはDSKに関する定義が書かれているヘッダファイル`c6x11dsk.h`の中で，`0x90080000`に定義されています．ここでは，DSKボードのユーザーsw1とユーザーLED1を使います．これらはいずれも第24ビット目に対応しています．このビットに0を書き込むとLEDは点灯し，1を書き込むと消灯します．

プログラムでは，メンバ関数`ReadRdy()`でアナログ信号を読み込んだ後，ユーザーsw1がON(値が1)のときはLEDを点灯し，左右のチャネルの差を求めます．一方，ユーザーsw1がOFF(値が0)のときはLEDを消灯し，左右のチャネルの和を求めます．最後に，求めたものを左右の両チャネルにメンバ関数`Write()`で出力します．

ビルドの際には，その他に割り込みベクタを記述したファイルとリンカコマンドファイルが必要になりますが，これらは**リスト2**，**リスト3**と同じものを使います．

注11：たとえば，McBSP1の受信割り込みであれば`0xF`という具合に，割り込み要因ごとにあらかじめ定義されている．

3. オーディオ用ドータカードを使うための基本的なプログラムの開発(割り込みの場合)

割り込みを使う場合は，それに対応するDSPの設定が必要になるので，そのためのクラスを作り，このクラスを使ったプログラムの例を示します．

3.1 割り込み方式で使用するクラス

割り込みのためのクラスは2.1で作成したクラス`PCM3003`を継承し，新たに割り込みの設定に必要な部分を追加した派生クラス`PCM3003Intr`を作成し，`PCM3003_intr.cpp`という名前を付けます．これを**リスト6**に示します．

● PCM3003Intr( )

派生クラス`PCM3003Intr`のコンストラクタでは，基本クラス`PCM3003`のコンストラクタへ引き数を渡し，さらにその時点での割り込み要求をすべてクリアします．このコンストラクタの引き数は基本クラス`PCM3003`のコンストラクタに渡され，2次キャッシュのサイズを設定します．引き数を与えない場合は，デフォルトの引き数の定義により，2次キャッシュのサイズは可能な最大の大きさ(64Kバイト)に設定されます．

● IntrGEnable( )

マスク可能な割り込みを許可します．

● IntrSetIER_IMUX( )

特定のCPU割り込み要因をCPU割り込みに割り当て，その割り込みを有効にします．書式は次のようになります．

`void IntrSetIER_IMUX(int INTSEL, int INTn)`

`INTSEL`　割り込み選択番号[注11]

〔**リスト5**〕**ボーカルを含む音楽からボーカルを除去するプログラム**(`VocalCanceler.cpp`)

```
//----------------------------------
//  Cancelation of center vocal part
//----------------------------------
#include "PCM3003 Polling.cpp"

int main()
{
    PCM3003 codec;
    short ch0, ch1;
    int sw;

    while (1)    // endless loop
    {
        codec.ReadRdy(ch0, ch1);                    // input

        sw = 0x01000000 & (*(u v int *)IO PORT);    // Read user sw1
        if (sw != 0)                                // user sw1 ON
        {
            *(u v int *)IO PORT = 0x06000000;       // user LED1 ON
            ch0 = ch0 - ch1;                        // Cancel vocal
        }
        else                                        // user sw1 OFF
        {
            *(u v int *)IO PORT = 0x07000000;       // user LED1 OFF
            ch0 = ch0 + ch1;                        // to monaural signal
        }
        ch1 = ch0;

        codec.Write(ch0, ch1);                      // output
    }
}
```

〔リスト6〕割り込みを使うためのクラス(PCM3003_intr.cpp)

```
//----------------------------------------------
//  class for interrupt on 'C6211/C6711 DSK
//      with PCM3003 daughter card
//          by MIKAMI, Naoki   2002/12/09
//----------------------------------------------
#include "PCM3003_Polling.cpp"

#ifndef MK_PCM3003_intr

class PCM3003Intr : public PCM3003
{
private:
    inline unsigned int McBSP1Read();
public:
    PCM3003Intr(banks_num banks = cach_bk4) : PCM3003(banks)
                     { ICR = 0xffff; }       // Clear all pending interrupts
    void IntrGEnable() { CSR |= 0x1; }       // Global interrupt enable
    void IntrSetIER_IMUX(int INTSEL, int INTn);
    inline void Read(short &ch0, short &ch1);
    inline void Read(float &ch0, float &ch1);
    inline void Read(short ch[]);
    inline void Read(float ch[]);

};

// Set intr. enable reg. and intr. multiplexer
//     INTSEL: Interrupt selection number
//     INTn  : Number of CPU Interrupt name
void PCM3003Intr::IntrSetIER_IMUX(int INTSEL, int INTn)
{
    int shift;
    if (INTn <= 9)
    {
        shift = (INTn - 4)*5;
        if (INTn>6) shift++;
        *(u_v_int *)IML = (*(u_v_int *)IML & ~(0xF<<shift) ) | INTSEL<<shift;
    }
    else
    {
        shift = (INTn - 10)*5;
        if (INTn>12) shift++;
        *(u_v_int *)IMH = (*(u_v_int *)IMH & ~(0xF<<shift) ) | INTSEL<<shift;
    }
    IER |= 0x2 | (1<<INTn);  // append to intr. enable register
}

// Receive data from PCM3003 using interrupt (data type: short)
inline void PCM3003Intr::Read(short &ch0, short &ch1)
{
    xn.buf = McBSP1Read();
    ch0 = xn.chn[0];
    ch1 = xn.chn[1];
}

// Receive normalized data from PCM3003 using interrupt (data type: float)
//      -1.0 <= data < 1.0
inline void PCM3003Intr::Read(float &ch0, float &ch1)
{
    xn.buf = McBSP1Read();
    ch0 = (float)xn.chn[0]*3.051758e-5f;
    ch1 = (float)xn.chn[1]*3.051758e-5f;
}

// Receive data from PCM3003 (data type: short[])
inline void PCM3003Intr::Read(short ch[])
{
    xn.buf = McBSP1Read();
    for (int k=0; k<2; k++) ch[k] = xn.chn[k];
}

// Receive normalized data from PCM3003 (data type: float[])
//      -1.0 <= data < 1.0
inline void PCM3003Intr::Read(float ch[])
{
    xn.buf = McBSP1Read();
    for (int k=0; k<2; k++) ch[k] = (float)xn.chn[k]*3.051758e-5f;
}

// Receive data from CODEC using McBSP1 (without ready flag check)
inline unsigned int PCM3003Intr::McBSP1Read()
{
    return (*(u_v_int *)McBSP1_DRR);
}

#define MK_PCM3003_intr
#endif
```

〔リスト7〕リセットベクタおよび割り込みベクタ(vec_Reset_INT5.asm)

```
;**************************************************************************
;  Vectors for Reset and INT5 (McBSP1 receive interrupt is mapped to INT5)  *
;       ---- for CCS 2.1 ----                                               *
;**************************************************************************
        .sect "vectors"
        .ref    _c_int00               ; C/C++ entry point
        .ref    _McBSP1_RX_ISR__Fv     ; Entry point of McBSP1 RX intr. service routine

RESET:                                 ; reset vector: address = 0x00000000
        MVKL    _c_int00,B0            ; B0: start address of _c_int00
        MVKH    _c_int00,B0
        B       B0                     ; branch to address pointed by B0
        NOP
        NOP
        NOP
        NOP
        NOP

        .space 0x20*4                  ; Reserve 0x80 bytes

INT5:                                  ; INT5 vector: address = 0x000000A0
        STW     A0,*B15--[1]           ; push A0
        MVKL    _McBSP1_RX_ISR__Fv,A0  ; A0: start address of _McBSP1_RX_ISR__Fv
        MVKH    _McBSP1_RX_ISR__Fv,A0
        B       A0                     ; branch to McBSP1 RX intr. service routine
        LDW     *++B15[1],A0           ; pop A0
        NOP
        NOP
        NOP     2
```

INTn　　CPU割り込みの番号

● Read()

McBSPの受信割り込みを使ってA-D変換器のデータを読み込む場合は，読み込み可能かどうかを調べる必要はないので，直ちにデータを読み込むためのメンバ関数を作成しました．McBSP1からデータを取り込むために，privateメンバ関数McBSP1Read()を使っています．

メンバ関数Read()はReadRdy()と同様に，4種類の引き数に対応するように多重定義しています．

3.2 割り込み方式で使用するクラスの使用例

割り込み処理に対応するプログラムを作成する際には，C/C++言語で書かれたソースプログラムのほかに，割り込みベクタをアセンブリ言語で記述したファイルも作成する必要があります．

(1) リセットベクタ，割り込みベクタ用ファイル

2.2ではリセットベクタのみを記述したファイルを作りましたが，ここではさらに割り込みベクタに対応する部分も記述する必要があります．**リスト7**に，リセットベクタおよび割り込みベクタ用ファイル(vec_Reset_INT5.asm)を示します．

このときに注意しなければならない点は，アセンブリ言語で書かれたプログラムからC++言語で書かれた関数名を参照するときの名前の書き方です．C言語で書かれている場合は，定義されている名前の先頭にアンダースコア"_"を付けることで，アセンブリ言語側から参照することができます．一方，C++言語の場合は関数の多重定義が許されるので，同じ関数名であっても，引き数により対応するものが異なってきます．

そこで，アセンブリ言語側から参照する場合はC++言語で定義されている名前の先頭にアンダースコア"_"を付けるほかに，関数名の後に二つのアンダースコア"__"を付け，さらに引き数の型や個数により決定される文字列を付ける必要があります．割り込みに対応する処理を記述する関数の場合は，引き数をもちません．その場合は"Fv"という文字列を付けます[注12]．次に示すC++言語のソースファイルでは，割り込み処理に対応する関数はMcBSP1_RX_ISR()になるので，アセンブリ言語側でこれを参照する場合には，_McBSP1_RX_ISR__Fv[注13]になります．

McBSP1の受信割り込みはINT5を使うようにします．そのため，対応するアドレスに，McBSP1_RX_ISR()を実行するための処理を書きます．

(2) C++言語によるソースプログラム

ここで作るプログラムは，**リスト4**に示した，PCM3003のA-D変換器から入力された信号を，何も処理を行わずにそのまま

注12：マニュアル(4)の第7章p.32によると，関数の引き数がある場合は，int型で一つの引き数をもつ関数の場合に"Fi"になるという記述があるが，それ以外の場合にこの文字列がどのようなものになるのかということは明記されていない．もし知りたい場合は，コンパイラが生成したアセンブリ言語のソースプログラムから知ることができる．

注13：コンパイラのバージョンにより命名規則は異なるようで，CCS1.23に付属のC++コンパイラの場合は，先頭のアンダースコアが二つ，つまり__McBSP1_RX_ISR__Fvになる．コンパイラのバージョンが異なる場合は，このように命名の規則が変わる可能性があるので，念のためコンパイラが生成したアセンブリ言語のソースプログラムを調べたほうがよい．なお，通常はコンパイラにより生成されるアセンブリ言語のファイルは，ビルド終了後に消去されるので，消去されないようにするにはAppendixのAssemblyオプションに示すように設定する必要がある．

〔**リスト8**〕**割り込み方式でA-D変換器からの入力信号をそのままD-A変換器に出力するプログラム**（Stereo_Thruogh_Int.cpp）

```
//--------------------------------------------------
//  Stereo signal through program using interrupt
//--------------------------------------------------
#include "PCM3003 intr.cpp"

PCM3003Intr codec;

int main()
{
    codec.IntrSetIER IMUX(0xF, 5);  // Assign RINT1 to INT5 and enable
    codec.IntrGEnable();            // Global interrupt enable

    while (1) {}                    // endless loop
}

// Interrupt service routine for McBSP1 receive
interrupt void McBSP1 RX ISR()
{
    float ch[2];

    codec.Read(ch);
//-----------------------------------------------
//  Put digital signal processing routine here
//-----------------------------------------------
    codec.Write(ch);
}
```

PCM3003のD-A変換器に送り出すという処理を，割り込みを使って実現するように変更したものです．これを**リスト8**に示します．

割り込みを使う場合は，“PCM3003_intr.cpp”をインクルードします．main()関数では，最初にメンバ関数IntrSetIER_IMUX()を使い，McBSP1の受信割り込み[注14]をCPU割り込みのINT5に割り当て，この割り込みを有効にします．次に，メンバ関数IntrGEnable()によりマスク可能な割り込みを許可します．その後，while(1)による無限ループに入りMcBSP1の受信割り込みを待ちます．

割り込みに対応する処理は関数の形で書き，関数名の先頭にinterruptというキーワードをつけます．受信割り込みを使ってA-D変換器から送られてきた値の入力を行う場合は，読み込み可能かどうかを調べる必要はありません．そこで，**リスト4**の場合とは異なり，メンバ関数Read()を使います．また，データの受け渡しは**リスト4**と同じでもかまわないのですが，ここでは使い方の例を示すためにshort型の配列を使っています．

*　　　　　　*

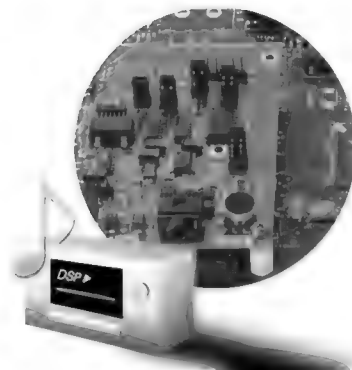
今回は，基本的なプログラミングについて解説を行いました．次回は，これを使ったアプリケーションのプログラミングについて解説を行います．

参考文献

1) 三上直樹，『C言語によるディジタル信号処理入門』，CQ出版（株），2002年
2) “PCM3002 PCM3003 16-/20-Bit Single-Ended Analog Input/Output STEREO AUDIO CODECs”，文献番号SBAS079，Burr-Brown，2000年
3) “Audio Daughter Card - TMDX326040A User Guide”，Revision 1.0，Texas Instruments（この文献はftp://ftp.ti.com/pub/cs/c6000/DSK/AudioDC/よりダウンロード可能）
4) 『TMS320C6000オプティマイジング（最適化）C/C++コンパイラ　ユーザーズ・マニュアル』，第2版，文献番号SPRU419A，日本テキサスインスツルメンツ，2001年（この文献は文献番号SPRU187Iの日本語訳で，http://www.tij.co.jp/jsc/docs/dsps/support/download/tools/よりダウンロード可能）
5) 『TMS320C6000ペリフェラルズ　リファレンス・ガイド』，第11章，日本テキサスインスツルメンツ，文献番号SPUR537,2001年（この文献は文献番号SPRU190Cの日本語訳で，http://www.tij.co.jp/jsc/docs/dsps/support/download/c6000/よりダウンロード可能）

みかみ・なおき　職業能力開発総合大学校　情報工学科

注14：割り込み要因に対応する番号はあらかじめ決められている．McBPS1の受信割り込みには0xFという番号が割り当てられている．

Appendix
Code Composer Studio Ver.2.10 について

本稿で使っているCode Composer Studio（CCS）のバージョンは2.1です（以下，CCS2.1）．そこで，CCS2.1を使う際のオプションの設定などについて簡単に説明します．

1．ビルド（Build）の際のオプション

● デバッグ用とリリース用の切り替え

CCS2.1では，ビルドにより2種類の実行コード（*.out）が生成されます．一つはデバッグ（Debug）用で，もう一つはリリース（Release）用のコードです．CCS2.1を立ち上げた直後のデフォルト状態ではデバッグ用の実行コードが生成されるように設定されています．

これをリリース用に切り替えるには，次のようにします．ツールバーの“Debug”と表示されているドロップダウンリストボックスをクリックすると，図Cに示すように選択可能なリストが現れるので，“Release”をクリックすれば，リリース用の実行コードを生成する設定になります．

デバッグ用に設定すると，ビルドの際には最適化が行われません．一方，リリース用に設定すると，ビルドの際にレベル3の最適化（もっとも高い最適化）が行われます．もちろん，これらはデフォルトでの話で，最適化のレベルはユーザーが自由に選択することができます．

●“Compiler”に関するオプション

ビルドの際のオプションを設定するためのウィンドウを開くには，図Dに示すようにメニューバーから［Project ｜ Build Options...］を選択します．そうすると，“Compiler”に関する設定が可能な状態になります．

▶ Basicオプション

図Eには，“Category:”の“Basic”の項目で，“Target Version:”を“671x”に設定した状態を示します．C6711 DSKを使用する際は，このように設定します．

この同じ画面で，ビルドの際に行われる最適化の設定も行うことができます．図Eで“Opt Level:”の項目を見ると，“None”となっています．これはデバッグ用のコードを生成する際のデフォルトです．この項目を“File（-o3）”に設定すれば，もっとも実行効率のよいコードを生成します．なお，リリース用のコードを生成する際のこの項目は，デフォルトで“File（-o3）”になっています．

▶ Preprocessorオプション

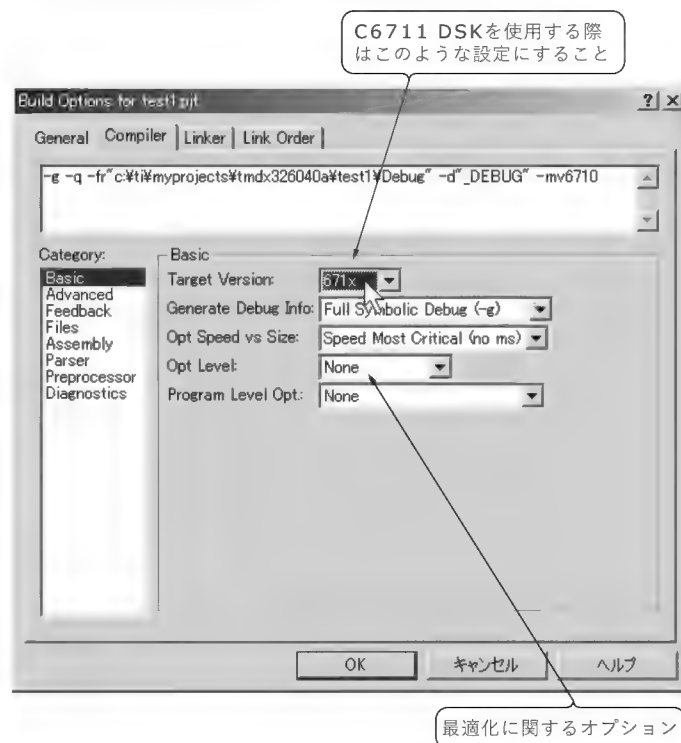
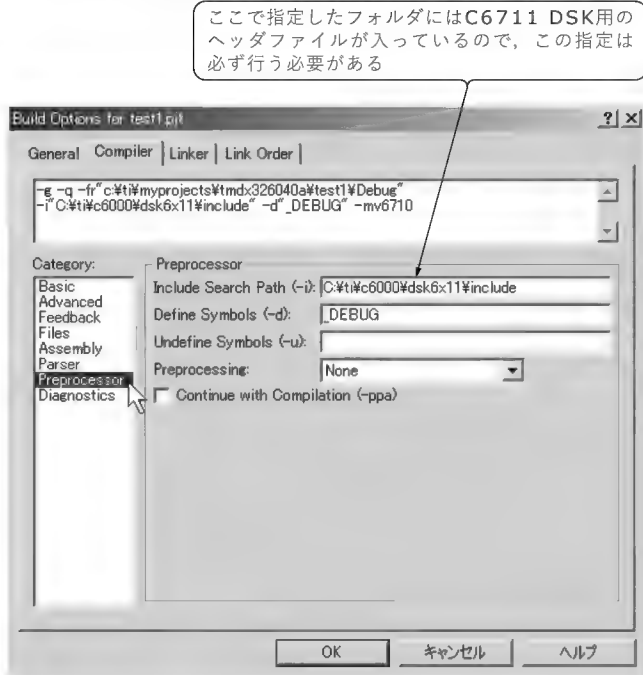
図Fに示すように，“Category:”の“Preprocessor”の項目を選択します．この画面で，“Include Search Path（-i）:”の項目を，この図に示すように設定します．ここで指定したフォルダにはC6711 DSK用のヘッダファイルが入っているので，この指定は必ず行う必要があります．その他，必要に応じてフォルダ名を追加します．

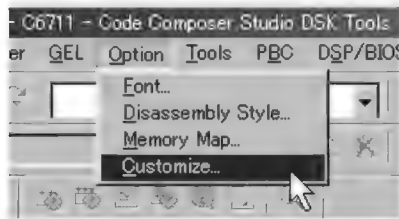
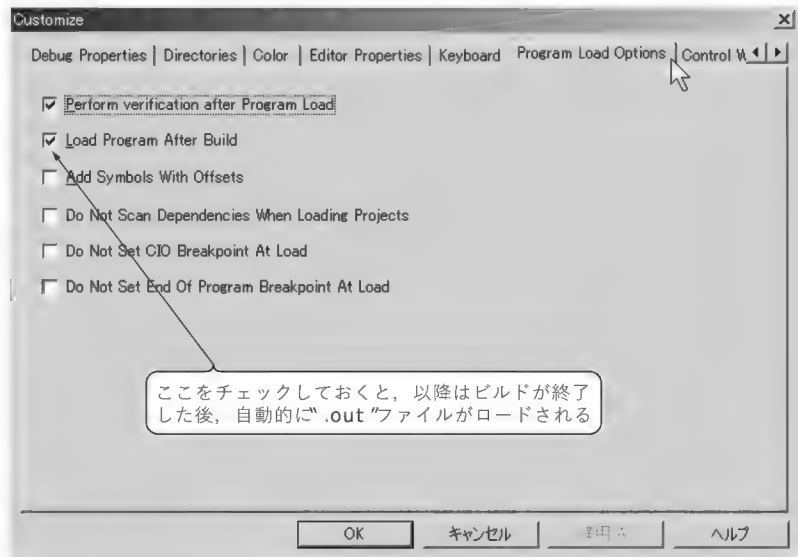
▶ Assemblyオプション

通常は，このオプションは特に設定する必要がありません．しかし，デフォルト状態では，コンパイラにより生成されたアセンブリ言語のファイルは，ビルド終了後自動的に消去されます．したがって，生成されたアセンブリ言語のファイルを見たい場合は，次のように設定す

〔図C〕デバッグ用/リリース用の設定

〔図D〕ビルドオプション設定ウィンドウを開く

〔図E〕CompilerのBasicオプション設定のようす

〔図F〕Compilerの Preprocessorオプションでインクルードファイルのパスを設定したようす

〔図G〕Compilerの Assemblyオプションでアセンブリファイルを残すための設定

〔図H〕ビルド完了後，実行形式のファイルを自動的にロードするためのオプション設定

る必要があります．

図Gに示すように，“Category:”の“Assembly”の項目を選択します．この画面で，“Keep generated .asm Files(-k):”の項目を，この図に示すように設定します．

2. 自動ロードオプション

ビルドが完了すると，実行形式のファイル(`*.out`)が生成されます．これを実行するためには，DSKボードへロードする必要があります．しかし，自動ロードのオプションを，有効になるように設定しておけば，実行形式のファイルが自動的にDSKボードへロードされるので，手動でロードする手間を省くことができます．その設定は，以下のように行います．

図Hの左側に示すように，メニューバーで[Option | Customize...]を選択すると，“Customize”ウィンドウが開きます．そこで，“Program Load Option”タブをクリックすると，右側のような画面が表示されます．この画面で，“Load Program After Build”のチェックボックスにチェックマークを付け，“OK”ボタンをクリックします．

この設定は，一度行えば，以降はCCSを再度立ち上げても有効な状態を保ちます．

第7回

C言語における GCCの拡張機能(2)

岸　哲夫

今回は，前回に引き続き，GNU Cで使用できる拡張機能について説明と検証を行う．各拡張機能は，どれも便利に使用できるというものではなく，可読性や可搬性をよく考えたうえで使う必要がある．本稿では，拡張機能によってもたらされる便利さと，可読性や可搬性などの点からの危険性を考慮し，それぞれの機能に対して考察する．

(編集部)

Red Hat Linuxの8.0がリリースされました．TurboLinuxも同じく8.0がリリースされました．他のディストリビュータも次々と新バージョンを出すことでしょう．

また，GCCの最新バージョンが3.2になり，STLの実装もかなり充実してきました．STLについては，C++言語の回で詳細に説明する予定なので，それまでお待ちください．

今回は，前回に続けてGNU Cの拡張機能について説明と検証を行います．

前回にも説明した複素数の取り扱いや長さが0の配列などについては，この連載の第3回(2002年10月号)で少し触れた「ISO/IEC 9899：1999-Programming Language C」(略称：C99)規格に含まれています．この規格については，拡張機能を説明した後に回を改めて説明します．

● 可変長自動配列

GNU Cでは可変長自動配列を宣言し，使うことができます．可変長自動配列の宣言と単純な自動配列との宣言の違いは，指定される長さが定数式ではないことです．記憶域は，配列が宣言されたところで割り当てられ，その宣言を含む構文が終了したところで解放されます．**リスト1～リスト4**に例を示します．

例はいささか実用性に欠けるソースですが，このような可変長配列が利用できます．生成されたアセンブラのAllocTblを見てください．可変長自動配列を実現させるためのコードが展開されています．

また，指定される配列の長さが定数でない場合でも，実行前に決定できる数値であれば最適化されます(**リスト5～リスト7**)．

リスト7を見るとわかるように，最適化によって可変長自動配列が最適化されて通常の配列に解釈されました．GCCの標準機能を使って同等の機能を実装するにはallocaを使うのが一般的だと思います．

通常のシステムでは，mallocを使って動的に領域を確保すると，その領域はヒープ領域に取られます．また，この領域を解放することを忘れると，システムに悪影響を及ぼします．allocaは標準ライブラリではありませんが，この関数はスタック領域に比較的高速に領域を確保し，確保した関数を抜けたら解放します．

可変長自動配列もスタック領域に領域を確保し，スコープの有効範囲を抜けたら解放します．

なお，可変長自動配列は，関数への引き数として使うことができます．

● 可変個数の引き数を受け取ることができるマクロ

```
#define eprintf(format, args...)
        fprintf (stderr, format , ## args)
```

のようなマクロを定義した場合，eprintf(arg1,arg2,arg3,arg4)もeprintf(arg1,arg2)も正しく解釈されます．

● 左辺値ではない配列の添字が存在

左辺値ではない配列に対して添字を使うことができます．ただし，単項演算子‘&’は使えません．このような構文を使用すると，可読性や可搬性に問題が起きるので推奨できません．

● voidポインタと関数へのポインタの算術演算

GNU Cでは，voidポインタや関数へのポインタで加算と減算の演算をすることが可能です．これは，voidや関数のサイズを1とみなして計算されます．この結果は，void型と関数型でさらにsizeofが許され，1を返すということです．

オプション-Wpointer-arithは，この拡張が使われた場合に警告を出します．

やはり通常は混乱の元になるので，voidならば想定される型にキャストするなりして，この拡張仕様は回避すべきです．

● 非定数による初期化

標準のCと同様に，GNU Cでも自動変数に割り当てられた集合体の初期化子の要素が，定数式である必要はありません(**リスト8～リスト12**)．

生成された関数tblを見るとわかるように，最適化されると，できるかぎり定数による初期化を試みます．生成されるコードは，定数による初期化のほうが効率よく高速に動作します．この記法は標準のCでも使われているので，可搬性に関しては問題ないと思います．場合に応じて使用してください．

● 生成関数式

GNU Cは，生成関数式「constructor expression」を使うこと

〔リスト1〕可変長自動配列を使った例(test48.c)

```
#include <stdio.h>
void     AllocTbl(int size);
int main(void)
{
    int  size;
    printf("input size ....:");
    scanf("%d",&size);
    AllocTbl(size);
    return;
}
void     AllocTbl(int size)
{
    char str[size];
    str[size+1]    =    0;
    return;
}
```

〔リスト2〕test48.cから生成されたアセンブラ(test48.s)

```
    .file     "test48.c"
    .version  "01.01"
gcc2 compiled.:
         .section  .rodata
.LC0:
    .string   "input size ....:"
.LC1:
    .string   "%d"
.text
    .align 4
.globl main
    .type      main,@function
main:
    pushl     %ebp
    movl      %esp, %ebp
    subl      $8, %esp
    subl      $12, %esp
    pushl     $.LC0
    call      printf
    addl      $16, %esp
    subl      $8, %esp
    leal      -4(%ebp), %eax
    pushl     %eax
    pushl     $.LC1
    call      scanf
    addl      $16, %esp
    subl      $12, %esp
    pushl     -4(%ebp)
    call      AllocTbl
    addl      $16, %esp
    leave
    ret
.Lfe1:
    .size      main,.Lfe1-main
    .align 4
.globl AllocTbl
    .type      AllocTbl,@function
AllocTbl:
    pushl     %ebp
    movl      %esp, %ebp
    pushl     %esi
    pushl     %ebx
    subl      $16, %esp
    movl      %esp, %esi
    movl      8(%ebp), %ebx
    decl      %ebx
    movl      %ebx, -16(%ebp)
    movl      $0, -12(%ebp)
    movl      $8, %eax
    mull      -16(%ebp)
    imull     $0, -16(%ebp), %ecx
    addl      %ecx, %edx
    imull     $8, -12(%ebp), %ecx
    addl      %ecx, %edx
    movl      %eax, %eax
    movl      %edx, %edx
    addl      $8, %eax
    adcl      $0, %edx
    leal      1(%ebx), %eax
    addl      $15, %eax
    shrl      $4, %eax
    movl      %eax, %eax
    sall      $4, %eax
    subl      %eax, %esp
    movl      %esp, %edx
    movl      8(%ebp), %eax
    incl      %eax
    movb      $0, (%eax,%edx)
    movl      %esi, %esp
    leal      -8(%ebp), %esp
    popl      %ebx
    popl      %esi
    popl      %ebp
    ret
.Lfe2:
    .size      AllocTbl,.Lfe2-AllocTbl
    .ident    "GCC: (GNU) 2.96 20000731 (Red Hat Linux 7.3 2.96-110)"
```

〔リスト3〕通常の自動配列を使った例(test49.c)

```
#include <stdio.h>
void     AllocTbl(int size);
int main(void)
{
    int  size;
    printf("input size but less than 10000....:");
    scanf("%d",&size);
    AllocTbl(size);
    return;
}
void     AllocTbl(int size)
{
    char str[10001];
    str[size+1]    =    0;
    return;
}
```

〔リスト4〕test49.cから生成されたアセンブラ(test49.s)

```
    .file     "test49.c"
    .version  "01.01"
gcc2 compiled.:
         .section  .rodata
    .align 32
.LC0:
    .string   "input size but less than 10000....:"
.LC1:
    .string   "%d"
.text
    .align 4
.globl main
    .type      main,@function
main:
    pushl     %ebp
    movl      %esp, %ebp
    subl      $8, %esp
    subl      $12, %esp
    pushl     $.LC0
    call      printf
    addl      $16, %esp
    subl      $8, %esp
    leal      -4(%ebp), %eax
    pushl     %eax
    pushl     $.LC1
    call      scanf
    addl      $16, %esp
    subl      $12, %esp
    pushl     -4(%ebp)
    call      AllocTbl
    addl      $16, %esp
    leave
    ret
.Lfe1:
    .size      main,.Lfe1-main
    .align 4
.globl AllocTbl
    .type      AllocTbl,@function
AllocTbl:
    pushl     %ebp
    movl      %esp, %ebp
    subl      $10024, %esp
    movl      8(%ebp), %eax
    incl      %eax
    leal      -10024(%ebp), %edx
    movl      %edx, %edx
    movb      $0, (%eax,%edx)
    leave
    ret
.Lfe2:
    .size      AllocTbl,.Lfe2-AllocTbl
    .ident    "GCC: (GNU) 2.96 20000731 (Red Hat Linux 7.3 2.96-110)"
```

〔リスト5〕可変長自動配列が最適化されて通常の配列に解釈される例(test50.c)

```
#include <stdio.h>
void     AllocTbl(int size);
int main(void)
{
    int  size;
    printf("input size ....:");
    scanf("%d",&size);
    AllocTbl(size);
    return;
}
void     AllocTbl(int size)
{
    int  size1    =    10;
    int  size2    =    20;
    char str[size1*size2];
    str[size+1]    =    0;
    return;
}
```

〔リスト6〕最適化なしのコンパイルで生成されたアセンブラ(test50.s)

```
	.file	"test50.c"
	.version	"01.01"
gcc2 compiled.:
		.section	.rodata
.LC0:
	.string	"input size ....:"
.LC1:
	.string	"%d"
.text
	.align 4
.globl main
	.type	main,@function
main:
	pushl	%ebp
	movl	%esp, %ebp
	subl	$8, %esp
	subl	$12, %esp
	pushl	$.LC0
	call	printf
	addl	$16, %esp
	subl	$8, %esp
	leal	-4(%ebp), %eax
	pushl	%eax
	pushl	$.LC1
	call	scanf
	addl	$16, %esp
	subl	$12, %esp
	pushl	-4(%ebp)
	call	AllocTbl
	addl	$16, %esp
	leave
	ret
.Lfe1:
	.size	main,.Lfe1-main
	.align 4
.globl AllocTbl
	.type	AllocTbl,@function
AllocTbl:
	pushl	%ebp
	movl	%esp, %ebp
	pushl	%esi
	pushl	%ebx
	subl	$16, %esp
	movl	%esp, %esi
	movl	$10, -12(%ebp)
	movl	$20, -16(%ebp)
	movl	-12(%ebp), %eax
	movl	%eax, %ebx
	imull	-16(%ebp), %ebx
	decl	%ebx
	movl	%ebx, -24(%ebp)
	movl	$0, -20(%ebp)
	movl	$8, %eax
	mull	-24(%ebp)
	imull	$0, -24(%ebp), %ecx
	addl	%ecx, %edx
	imull	$8, -20(%ebp), %ecx
	addl	%ecx, %edx
	movl	%eax, %eax
	movl	%edx, %edx
	addl	$8, %eax
	adcl	$0, %edx
	leal	1(%ebx), %eax
	addl	$15, %eax
	shrl	$4, %eax
	movl	%eax, %eax
	sall	$4, %eax
	subl	%eax, %esp
	movl	%esp, %edx
	movl	8(%ebp), %eax
	incl	%eax
	movb	$0, (%eax,%edx)
	movl	%esi, %esp
	leal	-8(%ebp), %esp
	popl	%ebx
	popl	%esi
	popl	%ebp
	ret
.Lfe2:
	.size	AllocTbl,.Lfe2-AllocTbl
	.ident	"GCC: (GNU) 2.96 20000731 (Red Hat Linux 7.3 2.96-110)"
```

〔リスト7〕最適化-O3オプションのコンパイルで生成されたアセンブラ(test51.s)

```
	.file	"test51.c"
	.version	"01.01"
gcc2 compiled.:
	.section	.rodata.str1.1,"aMS",@progbits,1
.LC0:
	.string	"input size ....:"
.LC1:
	.string	"%d"
.text
	.align 4
.globl AllocTbl
	.type	AllocTbl,@function
AllocTbl:
	pushl	%ebp
	movl	%esp, %ebp
	subl	$8, %esp
	movl	8(%ebp), %eax
	movl	%esp, %edx
	incl	%eax
	subl	$208, %esp
	movb	$0, (%esp,%eax)
	movl	%edx, %esp
	leave
	ret
.Lfe1:
	.size	AllocTbl,.Lfe1-AllocTbl
	.align 4
.globl main
	.type	main,@function
main:
	pushl	%ebp
	movl	%esp, %ebp
	subl	$20, %esp
	pushl	$.LC0
	call	printf
	popl	%edx
	popl	%ecx
	leal	-4(%ebp), %eax
	pushl	%eax
	pushl	$.LC1
	call	scanf
	popl	%eax
	pushl	-4(%ebp)
	call	AllocTbl
	addl	$16, %esp
	leave
	ret
.Lfe2:
	.size	main,.Lfe2-main
	.ident	"GCC: (GNU) 2.96 20000731 (Red Hat Linux 7.3 2.96-110)"
```

〔リスト8〕非定数による初期化例(test52.c)

```
#include <stdio.h>
void	Tbl(int param);
int main(void)
{
	int  data;
	printf("input tbl data ....:");
	scanf("%d",&data);
	Tbl(data);
	return;
}
void	Tbl(int param)
{
	int  tbl1[3]   =   {param,param*10,param*100};
	return;
}
```

〔リスト9〕test52.cから生成されたアセンブラ(test52.s)

```
	.file	"test52.c"
	.version	"01.01"
gcc2 compiled.:
.section .rodata
.LC0:
	.string	"input tbl data ....:"
.LC1:
	.string	"%d"
.text
	.align 4
.globl main
	.type	main,@function
main:
	pushl %ebp
	movl %esp,%ebp
	subl $24,%esp
	addl $-12,%esp
	pushl $.LC0
	call printf
	addl $16,%esp
	addl $-8,%esp
	leal -4(%ebp),%eax
	pushl %eax
	pushl $.LC1
	call scanf
	addl $16,%esp
	addl $-12,%esp
	movl -4(%ebp),%eax
	pushl %eax
	call Tbl
	addl $16,%esp
	jmp .L2
	.p2align 4,,7
.L2:
	movl %ebp,%esp
	popl %ebp
	ret
.Lfe1:
	.size	main,.Lfe1-main
	.align 4
.globl Tbl
	.type	Tbl,@function
Tbl:
	pushl %ebp
	movl %esp,%ebp
	subl $40,%esp
	movl 8(%ebp),%eax
	movl %eax,-24(%ebp)
	movl 8(%ebp),%edx
	movl %edx,%eax
	sall $2,%eax
	addl %edx,%eax
	leal (%eax,%eax),%edx
	movl %edx,-20(%ebp)
	movl 8(%ebp),%edx
	movl %edx,%eax
	sall $2,%eax
	addl %edx,%eax
	leal 0(,%eax,4),%edx
	addl %edx,%eax
	leal 0(,%eax,4),%edx
	movl %edx,-16(%ebp)
	movl -24(%ebp),%eax
	movl %eax,-12(%ebp)
	movl -20(%ebp),%eax
	movl %eax,-8(%ebp)
	movl -16(%ebp),%eax
	movl %eax,-4(%ebp)
	jmp .L3
	.p2align 4,,7
.L3:
	movl %ebp,%esp
	popl %ebp
	ret
.Lfe2:
	.size	Tbl,.Lfe2-Tbl
	.ident	"GCC: (GNU) 2.95.3 20010315 (release)"
```

〔リスト10〕非定数による初期化だが，最適化によって定数として扱われる例(test53.c)

```
#include <stdio.h>
void	Tbl(int param);
int main(void)
{
	Tbl(100);
	return;
}
void	Tbl(int param)
{
	int tbl1[3]	=	{param,param*10,param*100};
	return;
}
```

〔リスト11〕test53.cから最適化なしのオプションで生成されたアセンブラ(test53.s)

```
	.file	"test53.c"
	.version	"01.01"
gcc2 compiled.:
.text
	.align 4
.globl main
	.type	main,@function
main:
	pushl %ebp
	movl %esp,%ebp
	subl $8,%esp
	addl $-12,%esp
	pushl $100
	call Tbl
	addl $16,%esp
	jmp .L2
	.p2align 4,,7
.L2:
	movl %ebp,%esp
	popl %ebp
	ret
.Lfe1:
	.size	main,.Lfe1-main
	.align 4
.globl Tbl
	.type	Tbl,@function
Tbl:
	pushl %ebp
	movl %esp,%ebp
	subl $40,%esp
	movl 8(%ebp),%eax
	movl %eax,-24(%ebp)
	movl 8(%ebp),%edx
	movl %edx,%eax
	sall $2,%eax
	addl %edx,%eax
	leal (%eax,%eax),%edx
	movl %edx,-20(%ebp)
	movl 8(%ebp),%edx
	movl %edx,%eax
	sall $2,%eax
	addl %edx,%eax
	leal 0(,%eax,4),%edx
	addl %edx,%eax
	leal 0(,%eax,4),%edx
	movl %edx,-16(%ebp)
	movl -24(%ebp),%eax
	movl %eax,-12(%ebp)
	movl -20(%ebp),%eax
	movl %eax,-8(%ebp)
	movl -16(%ebp),%eax
	movl %eax,-4(%ebp)
	jmp .L3
	.p2align 4,,7
.L3:
	movl %ebp,%esp
	popl %ebp
	ret
.Lfe2:
	.size	Tbl,.Lfe2-Tbl
	.ident	"GCC: (GNU) 2.95.3 20010315 (release)"
```

〔リスト12〕test53.cから-O3オプションで最適化されて生成されたアセンブラ(test54.s)

```
	.file	"test54.c"
	.version	"01.01"
gcc2 compiled.:
.text
	.align 4
.globl Tbl
	.type	Tbl,@function
Tbl:
	pushl %ebp
	movl %esp,%ebp
	subl $24,%esp
	movl 8(%ebp),%eax
	movl %eax,-12(%ebp)
	leal (%eax,%eax,4),%eax
	leal (%eax,%eax),%edx
	movl %edx,-8(%ebp)
	leal (%eax,%eax,4),%eax
	sall $2,%eax
	movl %eax,-4(%ebp)
	movl %ebp,%esp
	popl %ebp
	ret
.Lfe1:
	.size	Tbl,.Lfe1-Tbl
	.align 4
.globl main
	.type	main,@function
main:
	pushl %ebp
	movl %esp,%ebp
	subl $8,%esp
	addl $-12,%esp
	pushl $100
	call Tbl
	movl %ebp,%esp
	popl %ebp
	ret
.Lfe2:
	.size	main,.Lfe2-main
	.ident	"GCC: (GNU) 2.95.3 20010315 (release)"
```

ができます．

GCCのマニュアルには，次のような記述があります．

宣言

```
struct foo {int a; char b[2];} structure;
```

生成関数式によってstruct fooを生成するコード(1)

```
structure = ((struct foo) {x + y, 'a', 0});
```

生成関数式によってstruct fooを生成するコード(2)

```
{
     struct foo temp = {x + y, 'a', 0};
     structure = temp;
}
```

GCCのマニュアルには，コード(1)とコード(2)は同等であると書かれていますが，実際には生成されたアセンブラを見るとわかるように，(1)のほうが少しコンパクトになります(**リスト13～リスト17**)．

結果は，コード(1)，すなわち生成関数式を使い最適化したもの，コード(1)で最適化しないもの，コード(2)すなわちANSI C準拠で書いたものの順でコンパクトになりました．

この機能を使って可搬性に問題が起こるとすれば，C++のクラスを使うべきだと思います．しかし，速度を上げ，実行形式も小さくしたいのであれば，この拡張機能を使用すべきでしょう．

● 配列の初期化について

ANSI Cでは，配列を初期化する際には，次のようにしなくてはなりません．

```
int tbl[5]={0,0,3,4,10};
```

しかし，拡張仕様では，次のような記述が認められます．

```
int tbl[5]={[4]=10,[2]3,[3]4};
```

すなわち，4番目の要素に10をセットし，2番目の要素に3をセットし3番目の要素に4をセットしています．

実際にコードを書いてみます(**リスト18**，**リスト19**)．

リストのようにANSI C形式で記述したものは，アセンブラ上でも0,0,3,4,10と配置するコードに生成されますが，この拡張仕様を使えば0が2要素，3,4,10と生成されています．

この拡張機能を使うとコンパクトかつ速いコードが生成されるはずです．

なお，**リスト20**，**リスト21**のような形式でも記述できます．生成されるアセンブラコードは同じですが，Cソースの可読性は高くなると思います．

〔**リスト13**〕**生成関数式によってstruct fooを生成するコード(1)の例**(test55.c)

```
#include <stdio.h>
struct foo {int a; char b[2];} structure;
void    test(int  x,int     y);
int main(void)
{
    int  x;
    int  y;
    scanf("%d",&x);
    scanf("%d",&y);
    test(x,y);
    return;
}
void    test(int  x,int     y)
{
    structure = ((struct foo) {x + y, 'a', 0});
}
```

〔**リスト14**〕**コード(1)で生成されたアセンブラ**(test55.s)

```
    .file     "test55.c"
    .version  "01.01"
gcc2 compiled.:
.section .rodata
.LC0:
    .string   "%d"
.text
    .align 4
.globl main
    .type      main,@function
main:
    pushl %ebp
    movl %esp,%ebp
    subl $24,%esp
    addl $-8,%esp
    leal -4(%ebp),%eax
    pushl %eax
    pushl $.LC0
    call scanf
    addl $16,%esp
    addl $-8,%esp
    leal -8(%ebp),%eax
    pushl %eax
    pushl $.LC0
    call scanf
    addl $16,%esp
    addl $-8,%esp
    movl -8(%ebp),%eax
    pushl %eax
    movl -4(%ebp),%eax
    pushl %eax
    call test
    addl $16,%esp
    jmp .L2
    .p2align 4,,7
.L2:
    movl %ebp,%esp
    popl %ebp
    ret
.Lfe1:
    .size      main,.Lfe1-main
    .align 4
.globl test
    .type      test,@function
test:
    pushl %ebp
    movl %esp,%ebp
    pushl %ebx
    movl 8(%ebp),%ecx
    movl 12(%ebp),%ebx
    addl %ebx,%ecx
    movl %ecx,%eax
    xorl %ecx,%ecx
    movb $97,%cl
    movw %cx,%dx
    movl %eax,structure
    movl %edx,structure+4
.L3:
    movl -4(%ebp),%ebx
    movl %ebp,%esp
    popl %ebp
    ret
.Lfe2:
    .size      test,.Lfe2-test
    .comm     structure,8,4
    .ident    "GCC: (GNU) 2.95.3 20010315 (release)"
```

〔リスト15〕最適化オプション-O3を付加してコード(1)で生成されたアセンブラ(test56.s)

```
    .file     "test56.c"
    .version  "01.01"
gcc2 compiled.:
.section .rodata
.LC0:
    .string   "%d"
    .comm     structure,8,4
.text
    .align 4
.globl test
    .type      test,@function
test:
    pushl %ebp
    movl %esp,%ebp
    movl 12(%ebp),%eax
    movl 8(%ebp),%edx
    addl %eax,%edx
    movl $97,%eax
    movw %ax,%cx
    movl %edx,structure
    movl %ecx,structure+4
    movl %ebp,%esp
    popl %ebp
    ret
.Lfe1:
    .size      test,.Lfe1-test
    .align 4
.globl main
    .type      main,@function
main:
    pushl %ebp
    movl %esp,%ebp
    subl $24,%esp
    addl $-8,%esp
    leal -4(%ebp),%eax
    pushl %eax
    pushl $.LC0
    call scanf
    addl $-8,%esp
    leal -8(%ebp),%eax
    pushl %eax
    pushl $.LC0
    call scanf
    movl -8(%ebp),%eax
    addl $32,%esp
    addl $-8,%esp
    pushl %eax
    movl -4(%ebp),%eax
    pushl %eax
    call test
    movl %ebp,%esp
    popl %ebp
    ret
.Lfe2:
    .size      main,.Lfe2-main
    .ident    "GCC: (GNU) 2.95.3 20010315 (release)"
```

〔リスト16〕生成関数式によってstruct fooを生成するコード(2)の例(test57.c)

```
#include <stdio.h>
struct foo {int a; char b[2];} structure;
void     test(int  x,int     y);
int main(void)
{
    int  x;
    int  y;
    scanf("%d",&x);
    scanf("%d",&y);
    test(x,y);
    return;
}
void     test(int  x,int     y)
{
    struct foo temp = {x + y, 'a', 0};
    structure = temp;
}
```

〔リスト17〕コード(2)で生成されたアセンブラ(test57.s)

```
    .file     "test57.c"
    .version  "01.01"
gcc2 compiled.:
.section .rodata
.LC0:
    .string   "%d"
.text
    .align 4
.globl main
    .type      main,@function
main:
    pushl %ebp
    movl %esp,%ebp
    subl $24,%esp
    addl $-8,%esp
    leal -4(%ebp),%eax
    pushl %eax
    pushl $.LC0
    call scanf
    addl $16,%esp
    addl $-8,%esp
    leal -8(%ebp),%eax
    pushl %eax
    pushl $.LC0
    call scanf
    addl $16,%esp
    addl $-8,%esp
    movl -8(%ebp),%eax
    pushl %eax
    movl -4(%ebp),%eax
    pushl %eax
    call test
    addl $16,%esp
    jmp .L2
    .p2align 4,,7
.L2:
    movl %ebp,%esp
    popl %ebp
    ret
.Lfe1:
    .size      main,.Lfe1-main
    .align 4
.globl test
    .type      test,@function
test:
    pushl %ebp
    movl %esp,%ebp
    subl $20,%esp
    pushl %ebx
    movl 8(%ebp),%ecx
    movl 12(%ebp),%ebx
    addl %ebx,%ecx
    movl %ecx,%eax
    xorl %ecx,%ecx
    movb $97,%cl
    movw %cx,%dx
    movl %eax,-8(%ebp)
    movl %edx,-4(%ebp)
    movl -8(%ebp),%eax
    movl -4(%ebp),%edx
    movl %eax,structure
    movl %edx,structure+4
.L3:
    movl -24(%ebp),%ebx
    movl %ebp,%esp
    popl %ebp
    ret
.Lfe2:
    .size      test,.Lfe2-test
    .comm     structure,8,4
    .ident    "GCC: (GNU) 2.95.3 20010315 (release)"
```

〔リスト 18〕配列を初期化する例 1(test58.c)

```
#include <stdio.h>
int main(void)
{
    int  tbl1[10]  =    {[4]=10,[2]3,[3]4};
    int  tbl2[10]  =    {0,0,3,4,10};
    return;
}
```

〔リスト 20〕配列を初期化する例 2(test59.c)

```
#include <stdio.h>
int main(void)
{
    int  tbl1[20]  =    {[0 ... 9]=2,[10 ... 19]4};
    int  tbl2[20]  =
            {2,2,2,2,2,2,2,2,2,2,4,4,4,4,4,4,4,4,4,4};
    return;
}
```

〔リスト 19〕test58.c から生成されたアセンブラ(test58.s)

```
    .file     "test58.c"
    .version  "01.01"
gcc2_compiled.:
.section .rodata
    .align 4
.LC0:
    .zero     8
    .long 3
    .long 4
    .long 10
    .zero     20
    .align 4
.LC1:
    .long 0
    .long 0
    .long 3
    .long 4
    .long 10
    .zero     20
.text
    .align 4
.globl main
    .type      main,@function
main:
    pushl %ebp
    movl %esp,%ebp
    subl $96,%esp
    pushl %edi
    pushl %esi
    leal -40(%ebp),%edi
    movl $.LC0,%esi
    cld
    movl $10,%ecx
    rep
    movsl
    leal -80(%ebp),%edi
    movl $.LC1,%esi
    cld
    movl $10,%ecx
    rep
    movsl
    jmp .L2
    .p2align 4,,7
.L2:
    leal -104(%ebp),%esp
    popl %esi
    popl %edi
    movl %ebp,%esp
    popl %ebp
    ret
.Lfe1:
    .size      main,.Lfe1-main
    .ident    "GCC: (GNU) 2.95.3 20010315 (release)"
```

〔リスト 21〕test59.c から生成されたアセンブラ(test59.s)

```
    .file     "test59.c"
    .version  "01.01"
gcc2_compiled.:
.section .rodata
    .align 4
.LC0:
    .long 2
    .long 2
    .long 2
    .long 2
    .long 2
    .long 2
    .long 2
    .long 2
    .long 2
    .long 2
    .long 4
    .long 4
    .long 4
    .long 4
    .long 4
    .long 4
    .long 4
    .long 4
    .long 4
    .long 4
    .align 4
.LC1:
    .long 2
    .long 2
    .long 2
    .long 2
    .long 2
    .long 2
    .long 2
    .long 2
    .long 2
    .long 2
    .long 4
    .long 4
    .long 4
    .long 4
    .long 4
    .long 4
    .long 4
    .long 4
    .long 4
    .long 4
.text
    .align 4
.globl main
    .type      main,@function
main:
    pushl %ebp
    movl %esp,%ebp
    subl $176,%esp
    pushl %edi
    pushl %esi
    leal -80(%ebp),%edi
    movl $.LC0,%esi
    cld
    movl $20,%ecx
    rep
    movsl
    leal -160(%ebp),%edi
    movl $.LC1,%esi
    cld
    movl $20,%ecx
    rep
    movsl
    popl %esi
    popl %edi
    movl %ebp,%esp
    popl %ebp
    ret
.Lfe1:
    .size      main,.Lfe1-main
    .ident    "GCC: (GNU) 2.95.3 20010315 (release)"
```

〔リスト22〕構造体の配列を初期化する例(test60.c)

```
#include <stdio.h>
struct CD_data1
{
char title[255];
char name[255];
char memo[255];
int Purchase_price;
};
struct CD_data2
{
char title[255];
char name[255];
char memo[255];
int Purchase_price;
};
struct CD_data3
{
char title[255];
char name[255];
char memo[255];
int Purchase_price;
};
int main(void)
{
    struct    CD_data1  mydata1 = {"hogehoge","hoge","(T_T)...",3000};
    struct    CD_data2  mydata2 = {name:"hoge",memo:"(T_T)...",Purchase_price:3000,title:"hogehoge"};
    struct    CD_data3  mydata3 = {.memo="(T_T)...",.Purchase_price=3000,title:"hogehoge",name:"hoge"};
    return;
}
```

〔リスト23〕test60.cから生成されたアセンブラ(test60.s)

```
    .file     "test60.c"
    .version  "01.01"
gcc2_compiled.:
.section .rodata
    .align 4
.LC0:
    .string   "hogehoge"
    .zero     246
    .string   "hoge"
    .zero     250
    .string   "memomemo"
    .zero     246
    .zero     3
    .long 3000
    .align 4
.LC1:
    .string   "hogehoge"
    .zero     246
    .string   "hoge"
    .zero     250
    .string   "memomemo"
    .zero     246
    .zero     3
    .long 3000
    .align 4
.LC2:
    .string   "hogehoge"
    .zero     246
    .string   "hoge"
    .zero     250
    .string   "memomemo"
    .zero     246
    .zero     3
    .long 3000
.text
    .align 4
.globl main
    .type      main,@function
main:
    pushl %ebp
    movl %esp,%ebp
    subl $2336,%esp
    pushl %edi
    pushl %esi
    leal -772(%ebp),%edi
    movl $.LC0,%esi
    cld
    movl $193,%ecx
    rep
    movsl
    leal -1544(%ebp),%edi
    movl $.LC1,%esi
    cld
    movl $193,%ecx
    rep
    movsl
    leal -2316(%ebp),%edi
    movl $.LC2,%esi
    cld
    movl $193,%ecx
    rep
    movsl
    popl %esi
    popl %edi
    movl %ebp,%esp
    popl %ebp
    ret
.Lfe1:
    .size      main,.Lfe1-main
    .ident    "GCC: (GNU) 2.95.3 20010315 (release)"
```

また，構造体の配列を初期化する際にも今までと違った方法でできます(**リスト22**，**リスト23**)．リストからわかるように，構造体中のどの要素に値をセットするかが明確になります．

なお，共用体の配列を初期化する際にも，**リスト24**に示すような，わかりやすい方法で行うことが可能です．

● case文で範囲指定を使用する

caseラベルにおいて連続した値の範囲を指定することができます．

```
case low ... high:
```

この例は，low以上high以下の個々の整数値について，個別にcaseラベルを記述するのと同等の効果をもちます．これは便利に使用できると思います．ANSI Cの仕様にこだわらないならば，可搬性を犠牲にして，可読性を取るのも良いと思います．

リスト25〜リスト28に示すように，case文で範囲指定を使用したほうが可読性も上がり，またアセンブラからわかるように速度も改善されると思います．

もっとも，**リスト29**のような条件の場合には，default文を使ったほうがコードは小さくなります．ただし，この記法では，dataの値が2,3,4,5の場合に何をするのかが一目ではわかりません．

〔リスト24〕共用体の配列を初期化する例(test61.c)

```
#include <stdio.h>
union data1
{
int i;
double  d;
};
union data2
{
int i;
double  d;
};
int main(void)
{
    union     data1     mydata1 = {d:128.02};
    union     data2     mydata2 = {i:128};
    return;
}
```

● 共用体へのキャスト

以下のような定義があるとします．

```
union foo { int i; double d; };
int x;
double y;,
```

この場合，

① u = (union foo) x

② u.i = x

〔リスト25〕case文で範囲指定を使用しない例(test62.c)

```
#include <stdio.h>
int main(void)
{
    int  temp;
    int  data;
    switch    (    data )
    {
        case 1:
            temp =    5;
        case 2:
            temp =    9;
        case 3:
            temp =    9;
        case 4:
            temp =    9;
        case 5:
            temp =    9;
        case 6:
            temp =    10;
    }
    return;
}
```

〔リスト26〕test62.cから生成されたアセンブラ(test62.s)

```
    .file    "test62.c"
    .version "01.01"
gcc2 compiled.:
.text
    .align 4
.globl main
    .type     main,@function
main:
    pushl %ebp
    movl %esp,%ebp
    subl $24,%esp
    movl -8(%ebp),%eax
    decl %eax
    cmpl $5,%eax
    ja .L3
    movl .L10(,%eax,4),%eax
    jmp *%eax
    .p2align 4,,7
.section .rodata
    .align 4
    .align 4
.L10:
    .long .L4
    .long .L5
    .long .L6
    .long .L7
    .long .L8
    .long .L9
.text
    .p2align 4,,7
.L4:
    movl $5,-4(%ebp)
.L5:
    movl $9,-4(%ebp)
.L6:
    movl $9,-4(%ebp)
.L7:
    movl $9,-4(%ebp)
.L8:
    movl $9,-4(%ebp)
.L9:
    movl $10,-4(%ebp)
.L11:
.L3:
    jmp .L2
    .p2align 4,,7
.L2:
    movl %ebp,%esp
    popl %ebp
    ret
.Lfe1:
    .size     main,.Lfe1-main
    .ident    "GCC: (GNU) 2.95.3 20010315 (release)"
```

〔リスト27〕
case文で範囲指定を使用する例(test63.c)

```
#include <stdio.h>
int main(void)
{
    int  temp;
    int  data;
    switch    (    data )
    {
        case 1:
            temp =    5;
        case 2 ... 5:
            temp =    9;
        case 6:
            temp =    10;
    }
    return;
}
```

〔リスト28〕test63.cから生成されたアセンブラ(test63.s)

```
    .file    "test63.c"
    .version "01.01"
gcc2 compiled.:
.text
    .align 4
.globl main
    .type     main,@function
main:
    pushl %ebp
    movl %esp,%ebp
    subl $24,%esp
    movl -8(%ebp),%eax
    cmpl $5,%eax
    jg .L9
    cmpl $2,%eax
    jge .L5
    cmpl $1,%eax
    je .L4
    jmp .L3
    .p2align 4,,7
.L9:
    cmpl $6,%eax
    je .L6
    jmp .L3
    .p2align 4,,7
.L4:
    movl $5,-4(%ebp)
.L5:
    movl $9,-4(%ebp)
.L6:
    movl $10,-4(%ebp)
.L8:
.L3:
    jmp .L2
    .p2align 4,,7
.L2:
    movl %ebp,%esp
    popl %ebp
    ret
.Lfe1:
    .size     main,.Lfe1-main
    .ident    "GCC: (GNU) 2.95.3 20010315 (release)"
```

〔リスト29〕
default文を使った例(test64.c)

```
#include <stdio.h>
int main(void)
{
    int  temp;
    int  data;
    switch    (    data )
    {
        case 1:
            temp =    5;
            break;
        case 6:
            temp =    10;
            break;
        default:
            temp =    9;
    }
    return;
}
```

〔リスト30〕test64.cから生成されたアセンブラ(test64.s)

```
    .file    "test64.c"
    .version "01.01"
gcc2 compiled.:
.text
    .align 4
.globl main
    .type     main,@function
main:
    pushl %ebp
    movl %esp,%ebp
    subl $24,%esp
    movl -8(%ebp),%eax
    cmpl $1,%eax
    je .L4
    cmpl $6,%eax
    je .L5
    jmp .L6
    .p2align 4,,7
.L4:
    movl $5,-4(%ebp)
    jmp .L3
    .p2align 4,,7
.L5:
    movl $10,-4(%ebp)
    jmp .L3
    .p2align 4,,7
.L6:
    movl $9,-4(%ebp)
.L3:
    jmp .L2
    .p2align 4,,7
.L2:
    movl %ebp,%esp
    popl %ebp
    ret
.Lfe1:
    .size     main,.Lfe1-main
    .ident    "GCC: (GNU) 2.95.3 20010315 (release)"
```

〔リスト31〕
ANCI C準拠のコードの例(test65.c)

```
#include <stdio.h>
union foo { int i; double d; };
int x;
double y;
int main(void)
{
    union foo u;
    x    =    10;
    u = (union foo) x;
    return;
}
```

〔リスト32〕**test65.cから生成されたアセンブラ**(test65.s)

```
    .file     "test65.c"
    .version  "01.01"
gcc2 compiled.:
.text
    .align 4
.globl main
    .type      main,@function
main:
    pushl %ebp
    movl %esp,%ebp
    subl $24,%esp
    movl $10,x
    movl x,%eax
    movl %eax,-8(%ebp)
    movl %edx,-4(%ebp)
    jmp .L2
    .p2align 4,,7
.L2:
    movl %ebp,%esp
    popl %ebp
    ret
.Lfe1:
    .size      main,.Lfe1-main
    .comm     x,4,4
    .comm     y,8,8
    .ident    "GCC: (GNU) 2.95.3 20010315 (release)"
```

③ u = (union foo) y

④ u.d = y

の①と②，③と④が等価になります．

共用体へのキャストを関数への引き数として使うことも可能です．この仕様は，生成関数式から派生しています．これを使うことは，可読性という面に関しては有益でしょう．

リスト31～リスト34に例を示します．アセンブラを比較すると，微妙にANSI C準拠のコードのほうがコンパクトになっています．共用体へのキャストを使用したほうのアセンブラで，1行増えた原因となっているコード，

movl %edx,-4(%ebp)

は不要であるような気がします．これは，最適化したら改善されました(**リスト35**)．

● 関数属性の宣言

GNU Cでは，関数のプロトタイプ宣言を定義する際に，属性を宣言することによって，コンパイラによる関数呼び出しの最適化をすることができるようになります．

キーワード__attribute__によって，宣言をする際に特別な属性を指定することができます．このキーワードの後に，二重の丸括弧(())に囲まれた属性指定が続きます．現在，9個の属性，

noreturn，const，format，
no_instrument_function，section，
constructor，destructor，unused，
weak

が関数に対して定義されています．sectionを含むその他の属性が，変数宣言(後述の変数属性の指定を参照)と型(同じく後述の型属性の指定を参照)に対してサポートされています．

個々のキーワードの前後に__を付けて属性を指定することもできるので，同じ名前をもつマクロが定義済みであっても，問

〔リスト33〕**共用体へのキャストを使用する例**(test66.c)

```
#include <stdio.h>
union foo { int i; double d; };
int x;
double y;
int main(void)
{
    union foo u;
    x    =    10;
    u.i = x;
    return;
}
```

〔リスト34〕**test66.cから生成されたアセンブラ**(test66.s)

```
    .file     "test66.c"
    .version  "01.01"
gcc2 compiled.:
.text
    .align 4
.globl main
    .type      main,@function
main:
    pushl %ebp
    movl %esp,%ebp
    subl $24,%esp
    movl $10,x
    movl x,%eax
    movl %eax,-8(%ebp)
    jmp .L2
    .p2align 4,,7
.L2:
    movl %ebp,%esp
    popl %ebp
    ret
.Lfe1:
    .size      main,.Lfe1-main
    .comm     x,4,4
    .comm     y,8,8
    .ident    "GCC: (GNU) 2.95.3 20010315 (release)"
```

〔リスト35〕**最適化してアセンブルした結果**(test67.s)

```
    .file     "test67.c"
    .version  "01.01"
gcc2 compiled.:
    .comm     x,4,4
    .comm     y,8,8
.text
    .align 4
.globl main
    .type      main,@function
main:
    pushl %ebp
    movl %esp,%ebp
    movl $10,x
    movl %ebp,%esp
    popl %ebp
    ret
.Lfe1:
    .size      main,.Lfe1-main
    .ident    "GCC: (GNU) 2.95.3 20010315 (release)"
```

題はありません．

○ **noreturn**

プログラム中で何らかの理由があり，呼び出し元に復帰しない関数が存在します．たとえば，関数中で異常終了させるような場合です．

このような関数をnoreturnと宣言することによって，その関数が復帰しないということをコンパイラに知らせることができます．**リスト36～リスト39**に例を示します．

なお，これ以降，本項の検証にあたっては，コンパイル時に

〔リスト36〕ANSI C準拠のコードの例(test68.c)

```
#include <stdio.h>
void abend_proc ();
int main(void)
{
    abend_proc();
    return;
}
void abend_proc ()
{
    printf("____abend_____      kita----");
    exit(1);
}
```

〔リスト38〕noreturn属性を使用する例(test69.c)

```
#include <stdio.h>
void abend_proc () __attribute__ ((noreturn));
int main(void)
{
    abend_proc();
    return;
}
void abend_proc ()
{
    printf("____abend_____      kita----");
    exit(1);
}
```

〔リスト39〕test69.cから生成されたアセンブラ(test69.s)

```
    .file     "test69.c"
    .version  "01.01"
gcc2 compiled.:
.section .rodata
.LC0:
    .string   "____abend_____      kita----"
.text
    .align 4
.globl abend_proc
    .type      abend_proc,@function
abend_proc:
    pushl %ebp
    movl %esp,%ebp
    subl $8,%esp
    addl $-12,%esp
    pushl $.LC0
    call printf
    addl $-12,%esp
    pushl $1
    call exit
.Lfe1:
    .size      abend_proc,.Lfe1-abend_proc
    .align 4
.globl main
    .type      main,@function
main:
    pushl %ebp
    movl %esp,%ebp
    subl $8,%esp
    call abend_proc
.Lfe2:
    .size      main,.Lfe2-main
    .ident    "GCC: (GNU) 2.95.3 20010315 (release)"
```

〔リスト37〕test68.cから生成されたアセンブラ(test68.s)

```
    .file     "test68.c"
    .version  "01.01"
gcc2 compiled.:
.section .rodata
.LC0:
    .string   "____abend_____      kita----"
.text
    .align 4
.globl abend_proc
    .type      abend_proc,@function
abend_proc:
    pushl %ebp
    movl %esp,%ebp
    subl $8,%esp
    addl $-12,%esp
    pushl $.LC0
    call printf
    addl $-12,%esp
    pushl $1
    call exit
.Lfe1:
    .size      abend_proc,.Lfe1-abend_proc
    .align 4
.globl main
    .type      main,@function
main:
    pushl %ebp
    movl %esp,%ebp
    subl $8,%esp
    call abend_proc
    movl %ebp,%esp
    popl %ebp
    ret
.Lfe2:
    .size      main,.Lfe2-main
    .ident    "GCC: (GNU) 2.95.3 20010315 (release)"
```

最適化オプション-O3を付加しています.

アセンブラからわかるように，noreturn属性は，abend_procが復帰することがないものと想定するようにコンパイラに指示します．コンパイラは，この場合には復帰することを考えず，復帰の後処理を省略しています．これにより，少しコンパクトなコードが生成されます．

なお，noreturn属性を指定した関数が，void以外の型の戻り値をもつのは無意味です．

noreturn属性は，バージョン2.5以前のGNU Cでは実装されていません．ちなみに，ある関数が復帰しないということを宣言するには，**リスト40**のような方法もあります．**リスト41**からわかるように，noreturn属性を使用したときとまったく同じように復帰の後処理を省略しています．

○ const

多くの関数は，引き数以外の値を参照しないでしょう．そして戻り値を返すこと以外に全体に影響を及ぼすこともないでしょう．

このような関数には，算術演算子と同様に，連載第2回(2002年9月号)で説明した共通部分式削除やループの最適化を適用することができます．このような関数は，const属性を指定して宣言するべきです．**リスト42～リスト45**に示す例ではkeisanという関数を何回か呼んでいますが，その結果は何度呼んでも変わらないので，アセンブラを見ると一度呼び出すだけで後は省略しています．

なお，const属性を指定した関数の戻り値がvoidになるのは無意味なことです．

○ format (archetype, string-index, first-to-check)

format属性は，その関数が，書式文字列に照らし合わせて型チェックを行うべき引き数，すなわちprintf，scanf，strftimeのような方式の引き数を取ることを指定します．

archetypeパラメータは，書式文字列がどのように解釈されるかを決定するものです．printf，scanf，strftimeのいず

〔リスト 40〕古い GNU C でも可能な定義例（test70.c）

```
#include <stdio.h>
typedef void voidfn ();
volatile voidfn abend_proc;
int main(void)
{
    abend_proc();
    return;
}
void abend_proc()
{
    printf("____abend_____      kita----");
    exit(1);
}
```

〔リスト 41〕test70.c から生成されたアセンブラ（test70.s）

```
    .file    "test70.c"
    .version "01.01"
gcc2_compiled.:
.section .rodata
.LC0:
    .string  "____abend_____      kita----"
.text
    .align 4
.globl abend_proc
    .type     abend_proc,@function
abend_proc:
    pushl %ebp
    movl %esp,%ebp
    subl $8,%esp
    addl $-12,%esp
    pushl $.LC0
    call printf
    addl $-12,%esp
    pushl $1
    call exit
.Lfe1:
    .size     abend_proc,.Lfe1-abend_proc
    .align 4
.globl main
    .type     main,@function
main:
    pushl %ebp
    movl %esp,%ebp
    subl $8,%esp
    call abend_proc
.Lfe2:
    .size     main,.Lfe2-main
    .ident    "GCC: (GNU) 2.95.3 20010315 (release)"
```

〔リスト 42〕ANSI C 準拠のコードの例（test71.c）

```
#include <stdio.h>
int keisan (int x);
int main(void)
{
    printf("%d",keisan(10));
    printf("%d",keisan(10));
    printf("%d",keisan(10));
    printf("%d",keisan(10));
    printf("%d",keisan(10));
    return;
}
int keisan(int x)
{
    return x * x;
}
```

〔リスト 43〕test71.c から生成されたアセンブラ（test71.s）

```
    .file    "test71.c"
    .version "01.01"
gcc2_compiled.:
.section .rodata
.LC0:
    .string  "%d"
.text
    .align 4
.globl keisan
    .type     keisan,@function
keisan:
    pushl %ebp
    movl %esp,%ebp
    movl 8(%ebp),%eax
    imull %eax,%eax
    movl %ebp,%esp
    popl %ebp
    ret
.Lfe1:
    .size     keisan,.Lfe1-keisan
    .align 4
.globl main
    .type     main,@function
main:
    pushl %ebp
    movl %esp,%ebp
    subl $8,%esp
    addl $-8,%esp
    addl $-12,%esp
    pushl $10
    call keisan
    pushl %eax
    pushl $.LC0
    call printf
    addl $32,%esp
    addl $-8,%esp
    addl $-12,%esp
    pushl $10
    call keisan
    pushl %eax
    pushl $.LC0
    call printf
    addl $32,%esp
    addl $-8,%esp
    addl $-12,%esp
    pushl $10
    call keisan
    pushl %eax
    pushl $.LC0
    call printf
    addl $32,%esp
    addl $-8,%esp
    addl $-12,%esp
    pushl $10
    call keisan
    pushl %eax
    pushl $.LC0
    call printf
    addl $32,%esp
    addl $-8,%esp
    addl $-12,%esp
    pushl $10
    call keisan
    pushl %eax
    pushl $.LC0
    call printf
    movl %ebp,%esp
    popl %ebp
    ret
.Lfe2:
    .size     main,.Lfe2-main
    .ident    "GCC: (GNU) 2.95.3 20010315 (release)"
```

〔リスト 44〕const 属性を使用する例（test72.c）

```
#include <stdio.h>
int keisan (int x) __attribute__ ((const));
int main(void)
{
    printf("%d",keisan(10));
    printf("%d",keisan(10));
    printf("%d",keisan(10));
    printf("%d",keisan(10));
    printf("%d",keisan(10));
    return;
}
int keisan(int x)
{
    return x * x;
}
```

〔リスト45〕**test72.cから生成されたアセンブラ2**(test72.s)

```
    .file    "test72.c"
    .version "01.01"
gcc2 compiled.:
.section .rodata
.LC0:
    .string   "%d"
.text
    .align 4
.globl keisan
    .type     keisan,@function
keisan:
    pushl %ebp
    movl %esp,%ebp
    movl 8(%ebp),%eax
    imull %eax,%eax
    movl %ebp,%esp
    popl %ebp
    ret
.Lfe1:
    .size     keisan,.Lfe1-keisan
    .align 4
.globl main
    .type     main,@function
main:
    pushl %ebp
    movl %esp,%ebp
    subl $20,%esp
    pushl %ebx
    addl $-8,%esp
    addl $-12,%esp
    pushl $10
    call keisan
    addl $16,%esp
    movl %eax,%ebx
    pushl %ebx
    pushl $.LC0
    call printf
    addl $-8,%esp
    pushl %ebx
    pushl $.LC0
    call printf
    addl $32,%esp
    addl $-8,%esp
    pushl %ebx
    pushl $.LC0
    call printf
    addl $-8,%esp
    pushl %ebx
    pushl $.LC0
    call printf
    addl $32,%esp
    addl $-8,%esp
    pushl %ebx
    pushl $.LC0
    call printf
    movl -24(%ebp),%ebx
    movl %ebp,%esp
    popl %ebp
    ret
.Lfe2:
    .size     main,.Lfe2-main
    .ident    "GCC: (GNU) 2.95.3 20010315 (release)"
```

れかでなければなりません．

string-indexパラメータは，どの引き数が書式文字列引き数であるかを数値で指定します．

また，first-to-checkは，書式文字列に照らし合わせてチェックすべき最初の引き数の番号です．チェックすべき引き数を指定できない関数に対しては，第3パラメータに0を指定してください．この場合，コンパイラは，書式文字列だけを対象にして整合性のチェックを行います．

○ **format_arg (string-index)**

format_arg属性は，その関数がprintfやscanfのような方式の引き数を取り，それを修正した後にprintfやscanfのような関数に渡すということを指定します．

〔リスト46〕**ANSI C準拠のコードの例**(test73.c)

```
#include <stdio.h>
int keisan (int x);
int keisan1 (int x);
int main(void)
{
    printf("%d",keisan(10));
    printf("%d",keisan(10));
    printf("%d",keisan(10));
    printf("%d",keisan(10));
    printf("%d",keisan(10));
    printf("%d",keisan1(10));
    printf("%d",keisan1(10));
    printf("%d",keisan1(10));
    printf("%d",keisan1(10));
    printf("%d",keisan1(10));
    return;
}
int keisan(int x)
{
    return x * x;
}
int keisan1(int x)
{
    return x * x;
}
```

○ **no_instrument_function**

オプション-finstrument-functionsを指定してコンパイルすると，ほとんどの関数の入口と出口において関数呼び出しのプロファイル処理を行うためのコードが生成されることになります．no_instrument_function属性をもつ関数については，そのようなコードの生成は行われません．

```
$ gcc -O3 -S -finstrument-functions test73.c
$ gcc -O3 -S -finstrument-functions test74.c
```

上のように**リスト46**，**リスト48**をコンパイルした場合，それぞれ**リスト47**，**リスト49**のようなコードが生成されます．const属性を付加した場合，関数呼び出しのプロファイル処理を行うためのコードが省略されています．

連載の第5回(2003年1月号)で説明していますが，-finstrument-functionsを指定すると，プロファイル用の関数が生成されます．

```
.globl __cyg_profile_func_enter
.globl __cyg_profile_func_exit
```

この関数を呼び出すためのコードが**リスト49**では省略されています．

○ **section ("section-name")**

通常，コンパイラは，生成したバイナリコードをtextセクションに置きます．しかし，ときには別のセクションを追加したり，特定の関数を特別なセクションに置く必要にせまられることがあります．section属性は，関数が特定のセクション内に置かれるよう指定します．

〔リスト47〕test73.cから生成されたアセンブラ(test73.s)

```
    .file     "test73.c"
    .version  "01.01"
gcc2_compiled.:
.globl __cyg_profile_func_enter
.globl __cyg_profile_func_exit
.section .rodata
.LC0:
    .string   "%d"
.text
    .align 4
.globl main
    .type      main,@function
main:
    pushl %ebp
    movl %esp,%ebp
    subl $20,%esp
    pushl %ebx
    movl 4(%ebp),%eax
    addl $-8,%esp
    pushl %eax
    pushl $main
    call __cyg_profile_func_enter
    addl $16,%esp
    addl $-8,%esp
    addl $-12,%esp
    pushl $10
    call keisan
    pushl %eax
    pushl $.LC0
    call printf
    addl $32,%esp
    addl $-8,%esp
    addl $-12,%esp
    pushl $10
    call keisan
    pushl %eax
    pushl $.LC0
    call printf
    addl $32,%esp
    addl $-8,%esp
    addl $-12,%esp
    pushl $10
    call keisan
    pushl %eax
    pushl $.LC0
    call printf
    addl $32,%esp
    addl $-8,%esp
    addl $-12,%esp
    pushl $10
    call keisan
    pushl %eax
    pushl $.LC0
    call printf
    addl $32,%esp
    addl $-8,%esp
    addl $-12,%esp
    pushl $10
    call keisan
    pushl %eax
    pushl $.LC0
    call printf
    addl $32,%esp
    addl $-8,%esp
    addl $-12,%esp
    pushl $10
    call keisan1
    pushl %eax
    pushl $.LC0
    call printf
    addl $32,%esp
    addl $-8,%esp
    addl $-12,%esp
    pushl $10
    call keisan1
    pushl %eax
    pushl $.LC0
    call printf
    addl $32,%esp
    addl $-8,%esp
    addl $-12,%esp
    pushl $10
    call keisan1
    pushl %eax
    pushl $.LC0
    call printf
    addl $32,%esp
    addl $-8,%esp
    addl $-12,%esp
    pushl $10
    call keisan1
    pushl %eax
    pushl $.LC0
    call printf
    addl $32,%esp
    addl $-8,%esp
    addl $-12,%esp
    pushl $10
    call keisan1
    pushl %eax
    pushl $.LC0
    call printf
    movl 4(%ebp),%eax
    addl $32,%esp
    addl $-8,%esp
    pushl %eax
    pushl $main
    call __cyg_profile_func_exit
    movl %ebx,%eax
    movl -24(%ebp),%ebx
    movl %ebp,%esp
    popl %ebp
    ret
.Lfe1:
    .size      main,.Lfe1-main
    .align 4
.globl keisan
    .type      keisan,@function
keisan:
    pushl %ebp
    movl %esp,%ebp
    subl $20,%esp
    pushl %ebx
    movl 8(%ebp),%ebx
    movl 4(%ebp),%eax
    addl $-8,%esp
    pushl %eax
    pushl $keisan
    call __cyg_profile_func_enter
    imull %ebx,%ebx
    movl 4(%ebp),%eax
    addl $16,%esp
    addl $-8,%esp
    pushl %eax
    pushl $keisan
    call __cyg_profile_func_exit
    movl %ebx,%eax
    movl -24(%ebp),%ebx
    movl %ebp,%esp
    popl %ebp
    ret
.Lfe2:
    .size      keisan,.Lfe2-keisan
    .align 4
.globl keisan1
    .type      keisan1,@function
keisan1:
    pushl %ebp
    movl %esp,%ebp
    subl $20,%esp
    pushl %ebx
    movl 8(%ebp),%ebx
    movl 4(%ebp),%eax
    addl $-8,%esp
    pushl %eax
    pushl $keisan1
    call __cyg_profile_func_enter
    imull %ebx,%ebx
    movl 4(%ebp),%eax
    addl $16,%esp
    addl $-8,%esp
    pushl %eax
    pushl $keisan1
    call __cyg_profile_func_exit
    movl %ebx,%eax
    movl -24(%ebp),%ebx
    movl %ebp,%esp
    popl %ebp
    ret
.Lfe3:
    .size      keisan1,.Lfe3-keisan1
    .ident    "GCC: (GNU) 2.95.3 20010315 (release)"
```

〔リスト48〕const属性を使用した例(test74.c)

```
#include <stdio.h>
int keisan (int x) __attribute__ ((no_instrument_function));
int keisan1 (int x);
int main(void)
{
    printf("%d",keisan(10));
    printf("%d",keisan(10));
    printf("%d",keisan(10));
    printf("%d",keisan(10));
    printf("%d",keisan(10));
    printf("%d",keisan1(10));
    printf("%d",keisan1(10));
    printf("%d",keisan1(10));
    printf("%d",keisan1(10));
    printf("%d",keisan1(10));
    return;
}
int keisan(int x)
{
    return x * x;
}
int keisan1(int x)
{
    return x * x;
}
```

〔リスト49〕test74.cから生成されたアセンブラ(test74.s)

```
    .file    "test74.c"
    .version "01.01"
gcc2 compiled.:
.globl __cyg_profile_func_enter
.globl __cyg_profile_func_exit
.section .rodata
.LC0:
    .string  "%d"
.text
    .align 4
.globl main
    .type     main,@function
main:
    pushl %ebp
    movl %esp,%ebp
    subl $20,%esp
    pushl %ebx
    movl 4(%ebp),%eax
    addl $-8,%esp
    pushl %eax
    pushl $main
    call __cyg_profile_func_enter
    addl $16,%esp
    addl $-8,%esp
    addl $-12,%esp
    pushl $10
    call keisan
    pushl %eax
    pushl $.LC0
    call printf
    addl $32,%esp
    addl $-8,%esp
    addl $-12,%esp
    pushl $10
    call keisan
    pushl %eax
    pushl $.LC0
    call printf
    addl $32,%esp
    addl $-8,%esp
    addl $-12,%esp
    pushl $10
    call keisan
    pushl %eax
    pushl $.LC0
    call printf
    addl $32,%esp
    addl $-8,%esp
    addl $-12,%esp
    pushl $10
    call keisan
    pushl %eax
    pushl $.LC0
    call printf
    addl $32,%esp
    addl $-8,%esp
    addl $-12,%esp
    pushl $10
    call keisan
    pushl %eax
    pushl $.LC0
    call printf
    addl $32,%esp
    addl $-8,%esp
    addl $-12,%esp
    pushl $10
    call keisan1
    pushl %eax
    pushl $.LC0
    call printf
    addl $32,%esp
    addl $-8,%esp
    addl $-12,%esp
    pushl $10
    call keisan1
    pushl %eax
    pushl $.LC0
    call printf
    addl $32,%esp
    addl $-8,%esp
    addl $-12,%esp
    pushl $10
    call keisan1
    pushl %eax
    pushl $.LC0
    call printf
    addl $32,%esp
    addl $-8,%esp
    addl $-12,%esp
    pushl $10
    call keisan1
    pushl %eax
    pushl $.LC0
    call printf
    addl $32,%esp
    addl $-8,%esp
    addl $-12,%esp
    pushl $10
    call keisan1
    pushl %eax
    pushl $.LC0
    call printf
    movl 4(%ebp),%eax
    addl $32,%esp
    addl $-8,%esp
    pushl %eax
    pushl $main
    call __cyg_profile_func_exit
    movl %ebx,%eax
    movl -24(%ebp),%ebx
    movl %ebp,%esp
    popl %ebp
    ret
.Lfe1:
    .size     main,.Lfe1-main
    .align 4
.globl keisan
    .type     keisan,@function
keisan:
    pushl %ebp
    movl %esp,%ebp
    movl 8(%ebp),%eax
    imull %eax,%eax
    movl %ebp,%esp
    popl %ebp
    ret
.Lfe2:
    .size     keisan,.Lfe2-keisan
    .align 4
.globl keisan1
    .type     keisan1,@function
keisan1:
    pushl %ebp
    movl %esp,%ebp
    subl $20,%esp
    pushl %ebx
    movl 8(%ebp),%ebx
    movl 4(%ebp),%eax
    addl $-8,%esp
    pushl %eax
    pushl $keisan1
    call __cyg_profile_func_enter
    imull %ebx,%ebx
    movl 4(%ebp),%eax
    addl $16,%esp
    addl $-8,%esp
    pushl %eax
    pushl $keisan1
    call __cyg_profile_func_exit
    movl %ebx,%eax
    movl -24(%ebp),%ebx
    movl %ebp,%esp
    popl %ebp
    ret
.Lfe3:
    .size     keisan1,.Lfe3-keisan1
    .ident    "GCC: (GNU) 2.95.3 20010315 (release)"
```

ただし，ファイル形式によっては，セクションの任意指定がサポートされていないものもあります．したがってsection属性は，すべてのプラットホームで利用できるわけではありません．あるモジュールのすべての内容を特定のセクションにマップする必要がある場合には，この属性ではなく，リンカの機能を使うことを検討してください．**リスト50**，**リスト51**に例を示します．

○ constructor

constructor属性を指定された関数は，main()関数が実行される前に，自動的に呼び出されるようになります．main()関数が実行される前に暗黙のうちに使われるデータを初期化するのに役に立ちます．**リスト52**，**リスト53**に例を示します．

コンパイルして実行した結果は次のとおりです．

```
$ gcc    -O3 -o test76 test76.c
```

〔リスト50〕section属性を使用した例(test75.c)

```
#include <stdio.h>
int keisan (int x)  __attribute__  ((section ("keisan")));
int keisan1 (int x)          __attribute__  ((section ("keisan1")));
int main(void)
{
    printf("%d",keisan(10));
    printf("%d",keisan1(10));
    return;
}
int keisan(int x)
{
    return x * x;
}
int keisan1(int x)
{
    return x * x;
}
```

〔リスト51〕test75.cから生成されたアセンブラ(test75.s)

```
    .file     "test75.c"
    .version  "01.01"
gcc2 compiled.:
.section .rodata
.LC0:
    .string   "%d"
.section keisan,"ax",@progbits
    .align 4
.globl keisan
    .type      keisan,@function
keisan:
    pushl %ebp
    movl %esp,%ebp
    movl 8(%ebp),%eax
    imull %eax,%eax
    movl %ebp,%esp
    popl %ebp
    ret
.Lfe1:
    .size      keisan,.Lfe1-keisan
.section keisan1,"ax",@progbits
    .align 4
.globl keisan1
    .type      keisan1,@function
keisan1:
    pushl %ebp
    movl %esp,%ebp
    movl 8(%ebp),%eax
    imull %eax,%eax
    movl %ebp,%esp
    popl %ebp
    ret
.Lfe2:
    .size      keisan1,.Lfe2-keisan1
.text
    .align 4
.globl main
    .type      main,@function
main:
    pushl %ebp
    movl %esp,%ebp
    subl $8,%esp
    addl $-8,%esp
    addl $-12,%esp
    pushl $10
    call keisan
    pushl %eax
    pushl $.LC0
    call printf
    addl $32,%esp
    addl $-8,%esp
    addl $-12,%esp
    pushl $10
    call keisan1
    pushl %eax
    pushl $.LC0
    call printf
    movl %ebp,%esp
    popl %ebp
    ret
.Lfe3:
    .size      main,.Lfe3-main
    .ident    "GCC: (GNU) 2.95.3 20010315 (release)"
```

```
$ ./test76
start
temp=10
```

リスト53に示す生成されたアセンブラには,

```
.section   .ctors,"aw"
```

というセクションができています. これはC++言語のコンストラクタそのものです.

○ **destructor**

destructor属性を指定された関数は, main()関数の実行

〔リスト52〕constructor属性を使用した例(test76.c)

```
#include <stdio.h>
void init proc ()     attribute   ((constructor ));
int temp =    100;
int main(void)
{
    printf("start\n");
    printf("temp=%d\n",temp);
    return;
}
void init proc()
{
    temp =    10;1
    return;
}
```

〔リスト53〕test76.cから生成されたアセンブラ(test76.s)

```
    .file     "test76.c"
    .version  "01.01"
gcc2 compiled.:
.globl temp
.data
    .align 4
    .type      temp,@object
    .size      temp,4
temp:
    .long 100
.section .rodata
.LC0:
    .string   "start\n"
.LC1:
    .string   "temp=%d\n"
.section .ctors,"aw"
    .long      init proc
.text
    .align 4
.globl init proc
    .type      init proc,@function
init proc:
    pushl %ebp
    movl %esp,%ebp
    movl $10,temp
    movl %ebp,%esp
    popl %ebp
    ret
.Lfe1:
    .size      init proc,.Lfe1-init proc
    .align 4
.globl main
    .type      main,@function
main:
    pushl %ebp
    movl %esp,%ebp
    subl $8,%esp
    addl $-12,%esp
    pushl $.LC0
    call printf
    movl temp,%eax
    addl $-8,%esp
    pushl %eax
    pushl $.LC1
    call printf
    movl %ebp,%esp
    popl %ebp
    ret
.Lfe2:
    .size      main,.Lfe2-main
    .ident    "GCC: (GNU) 2.95.3 20010315 (release)"
```

が終了した後に，自動的に呼び出されるようになります．この機能は実行後の後処理に役に立ちます．

リスト54に示すソースをコンパイルして実行した結果は，次のとおりです．

```
$ gcc    -O3 -o test77 test77.c
$ ./test77
program_start
main execute
program_end
```

リスト55に示す生成されたアセンブラには，

```
.section   .dtors,"aw"
```

というセクションができています．これはC++言語のデストラクタそのものです．

○ **unused**

この属性が関数に対して指定された場合，その関数は使われ

〔**リスト54**〕**destructor属性を使用した例**（test77.c）

```
#include <stdio.h>
void init proc ()  __attribute   ((constructor ));
void exit proc ()    attribute   ((destructor ));
int main(void)
{
    printf("main execute\n");
    return;
}
void init proc()
{
    printf("program start\n");
    return;
}
void exit proc()
{
    printf("program end\n");
    return;
}
```

ないはずであるという意味になります．GNU Cは，このような関数に対しては警告メッセージを出力しません．なお，C++ではパラメータをもたない定義は正当なので，現在のところGNU C++は，この属性をサポートしていません．

○ **weak**

weak属性を指定された宣言は，globalシンボルではなく，weakシンボルとして出力されるようになります．これは主として，ユーザーのソースコード中で無効にすることのできるライブラリ関数を定義するのに役に立ちますが，関数以外の宣言においても使うこともできます．このシンボルは，ELFターゲットにおいてサポートされていますが，GNUのアセンブラとリンカを使っている場合には，a.outターゲットにおいてもサポートされています．

リスト56と**リスト57**を照らし合わせればわかるようにinit_proc1はweakシンボルとして，init_procはグローバルシンボルとして生成されています．

生成された実行形式を調べると，次に示すようにinit_procの属性はTで，init_proc1の属性はWとなっています．

```
$ nm test78 | grep init
08048298 ? _init
080483e8 t init_dummy
08048458 t init_dummy
08048424 T init_proc
0804841c W init_proc1
```

ここで，Wは「弱い定義（weak）」であることを示します．すなわち，このシンボルは定義されてはいますが，他のライブラリの定義によって上書きされることを示しています．通常の定義は

〔**リスト55**〕**test77.cから生成されたアセンブラ**（test77.s）

```
    .file    "test77.c"
    .version  "01.01"
gcc2 compiled.:
.section .rodata
.LC0:
    .string   "main execute\n"
.LC1:
    .string   "program start\n"
.section .ctors,"aw"
    .long      init proc
.section .rodata
.LC2:
    .string   "program end\n"
.section .dtors,"aw"
    .long      exit proc
.text
    .align 4
.globl init proc
    .type      init proc,@function
init proc:
    pushl %ebp
    movl %esp,%ebp
    subl $8,%esp
    addl $-12,%esp
    pushl $.LC1
    call printf
    movl %ebp,%esp
    popl %ebp
    ret
.Lfe1:
    .size      init proc,.Lfe1-init proc
    .align 4
.globl exit proc
    .type      exit proc,@function
exit proc:
    pushl %ebp
    movl %esp,%ebp
    subl $8,%esp
    addl $-12,%esp
    pushl $.LC2
    call printf
    movl %ebp,%esp
    popl %ebp
    ret
.Lfe2:
    .size      exit proc,.Lfe2-exit proc
    .align 4
.globl main
    .type      main,@function
main:
    pushl %ebp
    movl %esp,%ebp
    subl $8,%esp
    addl $-12,%esp
    pushl $.LC0
    call printf
    movl %ebp,%esp
    popl %ebp
    ret
.Lfe3:
    .size      main,.Lfe3-main
    .ident    "GCC: (GNU) 2.95.3 20010315 (release)"
```

〔リスト56〕weak属性を使用した例(test78.c)

```
#include <stdio.h>
void init_proc1() __attribute__ ((weak   ));
void init_proc();
int main(void)
{
    printf("start\n");
    return;
}
void init_proc1()
{
}
void init_proc()
{
}
```

Tで示されます(init_proc).

新たにソースtest79.c(**リスト58**)を作り，その中でinit_proc1を再定義した場合，通常はリンクエラーとなりますが，weak属性が付いているので，新しいinit_proc1がリンクされます.

```
$ gcc    -o test78 test78.c test79.c
$ nm test78 | grep init
08048298 ? _init
080483e8 t init_dummy
08048478 t init_dummy
08048424 T init_proc
08048430 T init_proc1
```

これはデバッグ時に役立ちそうな機能です.

○ alias("target")

alias属性を指定された宣言は，別のシンボルへの別名として出力されます．その別のシンボルは指定されていなければなりません.

リスト59に示すソースをコンパイル/実行した例を次にあげます.

```
$ gcc    -o test81 test81.c
$ ./test81
start
ver1.0
ver1.0
```

その後，新しいinit_proc1のソース(**リスト60**)をリンクします．実行結果は，次のようになります.

```
$ gcc    -o test81 test81.c test80.c
$ ./test81
start
ver1.0
ver1.1
```

このように，関数init_proc1はtest80.cのものに置き換わりました.

これもやはりデバッグ時に役立ちそうな機能です.

○ no_check_memory_usage

コンパイル時のオプション-fcheck-memory-usageが指定

〔リスト57〕test78.cから生成されたアセンブラ(test78.s)

```
    .file    "test78.c"
    .version "01.01"
gcc2 compiled.:
.section .rodata
.LC0:
    .string  "start\n"
.text
    .align 4
.globl main
    .type     main,@function
main:
    pushl %ebp
    movl %esp,%ebp
    subl $8,%esp
    addl $-12,%esp
    pushl $.LC0
    call printf
    addl $16,%esp
    jmp .L2
    .p2align 4,,7
.L2:
    movl %ebp,%esp
    popl %ebp
    ret
.Lfe1:
    .size     main,.Lfe1-main
    .align 4
    .weak    init_proc1
    .type     init_proc1,@function
init_proc1:
    pushl %ebp
    movl %esp,%ebp
.L3:
    movl %ebp,%esp
    popl %ebp
    ret
.Lfe2:
    .size     init_proc1,.Lfe2-init_proc1
    .align 4
.globl init_proc
    .type     init_proc,@function
init_proc:
    pushl %ebp
    movl %esp,%ebp
.L4:
    movl %ebp,%esp
    popl %ebp
    ret
.Lfe3:
    .size     init_proc,.Lfe3-init_proc
    .ident   "GCC: (GNU) 2.95.3 20010315 (release)"
```

〔リスト58〕新しいinit_proc1のソース(test79.c)

```
#include <stdio.h>
void init_proc1();
void init_proc1()
{
    printf("init_proc1");
}
```

〔リスト59〕alias属性を付加したソース(test81.c)

```
#include <stdio.h>
void init_proc1() __attribute__ ((weak, alias
("init_proc")));
void init_proc();
int main(void)
{
    printf("start\n");
    init_proc();
    init_proc1();
    return;
}
void init_proc()
{
    printf("ver1.0\n");
}
```

〔リスト60〕新しいinit_proc1のソース（test80.c）

```
#include <stdio.h>
void init_proc1();
void init_proc1()
{
    printf("ver1.1\n");
}
```

されていると，メモリアクセスの前に，サポートルーチンの呼び出しを行うコードが生成されますが，関数にこの属性を指定すると，その関数については，メモリチェックを行うコードが無効化されます．

この-fcheck-memory-usageオプションを指定すると，メモリチェックが有効となっている関数の中では，asmキーワードや，__asm__キーワードは使えませんが，no_check_memory_usageを指定した関数内では使用できます．

連載第5回で，-fcheck-memory-usageオプションについて説明しています．

○ regparm (number)

Intel 386上では，regparm属性により，コンパイラは最高でnumberによって指定される個数までの整数引き数を，スタックではなくEAX，EDX，ECXレジスタに入れて渡すようになります．

○ stdcall

Intel 386上では，stdcall属性により，関数が可変個数の引き数を取るのでない限り，引き数を渡すのに使われたスタック領域は呼び出された関数がPOPするものと，コンパイラは想定するようになります．

○ cdecl

Intel 386上では，cdecl属性により，引き数を渡すのに使われたスタック領域は呼び出した関数がPOPするものと，コンパイラは想定するようになります．ここに挙げた3点は現在では古いアーキテクチャ固有のものであり，もう使うべきではないと思います．

○ longcall

RS/6000とPowerPC上では，longcall属性により，コンパイラは常にポインタを使ってその関数を呼び出すようになります．これにより，現在の位置から64Mバイトを超えて離れた位置にある関数でも呼び出すことができます．

dllimport属性，dllexport属性，exception属性はPowerPC上で動作するWindows NT固有のものなので省略します．

○ function_vector

このオプションは，H8/300とH8/300Hにおいて，指定された関数が関数ベクタを利用して呼び出されるべきであることを示すのに使います．

○ interrupt_handler

このオプションは，H8/300とH8/300Hにおいて，指定された関数が割り込みハンドラであることを示すのに使います．

○ eightbit_data

このオプションは，H8/300とH8/300Hにおいて，指定された変数が8ビットデータセクションに置かれるべきであることを示すのに使います．

○ tiny_data

このオプションは，H8/300Hにおいて，指定された変数がtinyデータセクションに置かれるべきであることを示すのに使います．

○ interrupt

このオプションは，M32R/Dにおいて，指定された関数が割り込みハンドラであることを示すのに使います．

○ model (model-name)

この属性は，M32R/Dにおいて，オブジェクトのアドレス範囲を設定し，関数に対して生成されるコードを決定するのに使います．

*　　　　　*

次回は続けてGNU　Cの拡張機能について，詳細に説明と検証を行います．

きし・てつお

IPパケットの隙間から

エラーメールから見える世相

祐安重夫

53

メーリングリストの管理というのは，規模にもよるが，けっこう面倒なものである．登録されていても到達しなくなるアドレスが，しばしば存在するからである．

筆者たちがやっている大海通信(http://www.daikai.imac.jp/)のような小規模でプライベートなものでも，連絡なしにメールアドレスを頻繁に変更する人が時々いて，メールが届かなくなり，エラーメールが返ってくる．メンバの一人であるMさんなど，もちろん冗談でだが，「祐安の陰謀で投稿できなくなった」などと言い出したりしていたが，最近は律義にアドレス変更のお知らせをしてくれるようになった．

数十人規模で公開されており，完全な第三者を含むものだと，連絡なしにメールアドレスが存在しなくなったりする例は，けっこうあるようだ．これが数百人規模となると，1回メールが配送されると，管理者宛のエラーメールが5通から10通やってくるようなケースも普通にある．月の変わり目，年度替わりなどをきっかけにどっと増えて，対処してもすぐにまたそのようなアドレスが出現する．

これが数千人から数万人規模のメールマガジンとなると，さらに問題は多くなる．うっかりするとエラーメールは，数百通などという状態に，簡単になってしまう．

もともと現在のインターネットの原型となったネットワークの多くは，特定少数のメンバの情報交換や資源の共有などを目的としてつくられた．メールアドレスも基本的にはユーザーが特定可能で，メールアドレスの変更にも比較的簡単に追従できた．それらのネットワークが相互に接続され，インターネットの初期の形態を確立し始めた頃も，人数は増加してもそれほど状況には変化はなかった．1970年代後半から，1980年代初頭にかけての話である．日本に現在のインターネットの原型ができたのは1980年代後半だが，そこでもユーザーは特定できる個人であり，日本から発信されるようになったfj.*のニュースグループでは，当初から実名主義をとっていた．

1990年代に入ると，商用のインターネットプロバイダが出現し，また既存のパソコン通信サービスもインターネットと接続するようになった．だが，パソコン通信ではハンドルネームを使用した半匿名という形態をとっていることが多く，その頃のfj.*では，習慣(AUP)の違いからくる揉め事が，けっこう起こっていたと記憶している．

現在では一人の個人が複数のメールアドレスをもつことは，きわめて当たり前のこととなっており，筆者にしても最終的に手元に転送されてくるアドレスは，たぶん50を超えている．これほど極端な数ではなくとも，目的別に複数の受信アドレスを使い分けているユーザーは，かなりいるはずである．そのうえ，現在ではhotmailなどのフリーメールアドレスをサービスするところがいくつもあり，ユーザーとしては完全に匿名性が保証された(と思い込まされている)アドレスを使用している人がかなりいる．

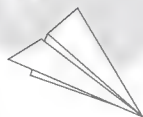
メーリングリストやメールマガジンに問題を戻すと，このようなメールアドレスをめぐる状況が，事態を複雑化させているのはたしかである．では管理者として具体的に，どのような問題があり，どう対処すればいいのだろうか．

第一のケースとして，User Unknownとなるアドレスについて考えてみよう．これについては，基本的に配送リストから外すのが正解であろう．もちろん，相手側メールサーバの一時的な障害の影響という場合もあるので，一応数回はようすを見たほうがいいだろう．このアドレスがフリーメールのものだと，こちらも躊躇なく配送リストから外すことができる．メーリングリストによっては，最初からフリーメールアドレスからの参加を拒否しているところもある．この場合，企業内のメールシステムとしてSMTPによらないものを使用しているケースなどでは，SMTPゲートウェイがRFCに準拠しない形式でエラーを返してくることがあり，判断に苦しむことがある．

インターネットとメールの交換をするなら，そのためのゲートウェイは，まともな仕様にしてほしいものだ．このケースでいちばん困るのは，User Unknownになったアドレスが，実際には登録されたアドレスではない場合だ．ようするに登録されたアドレスに，ユーザーがメールの転送設定をしているが，その転送先が存在しないという場合だ．これにはお手上げである．

第二のケースはHost Unknownとなる場合だ．サブドメインがそうなる場合は，相手側のDNSの設定などを疑うこともできるが，最近では組織内のメールシステムの全面的な変更などで，同じドメインのアドレスが一斉に届かなくなることがある．実際には移行の猶予期間があるはずだが，その間に連絡がないと配送リストから外すしかなくなる．

これがドメイン自体がHost Unknownとなる場合，どう処理すればいいのだろう．DNSの問題という場合もあり得るが，今回それについてwhoisデータベースでチェックを行っていったところ，とくにco.jpドメインで，ドメイン自体が消滅してしまっているものをかなり発見した．また，ドメイン自体は存在しているが，いつのまにかインターネットとの接続がなくなっているところもいくつか目についた．これは世の中の不況を反映したものなのだろうか．

すけやす・しげお　インターメディア・アクセス

● 予期できなかった恐るべき事件

自宅に戻り，いつものようにパソコンに電源を入れた．しかし，不思議なことにウンともスンともいわない．LEDさえも燈らない．これにはたいへん困ってしまった．メインのパソコンなので，メールもできない．第一，やり残している仕事の続きもできやしない．

数時間かけてどこが悪いのかを探しまくり，最終的に，もう1台のパソコンから電源ケーブルを引っ張り出して動くことを確認，電源ユニットが壊れていることが判明した．とりあえず，パソコン2台をケースの蓋を開けたまま並べ，電源ケーブルを繋ぎ換えて急場をしのぐことにした．

翌日，事務所に行き，ふだんあまり使われていない1台のパソコンから，電源ユニットをひとまずお借りすることにして，家に持って帰った．その時は，これが恐るべき事件の始まりだとは，まったく予想もつかなかった．

さっそく自宅に戻り，持ち帰った電源ユニットをいったん机の上に置いた．次いで，壊れたほうのパソコンの電源ユニットをはずした．さあ交換しようと，机の上の電源ユニットを再び振り返って見た時だ．筆者は完全に固まってしまった．その電源ユニットの背後のファンから，何か線のようなものが2本，飛び出しているではないか．しかも，その線はユラリユラリと揺れている．まるで昆虫の触覚のようだ．ボーッと眺めていると，頭が正体を現した．そして，ゆっくり全身が現れて，机に降り立った．周りのようすを気にするようすもなく，音も立てずに机の上をゆっくり歩いている．ただ眺めていただけの筆者は，踵を返して疾風のように台所へと向かい，台所洗剤をもってあっという間に部屋に帰ってきた．

はずだったが，すでにそこには敵は影すらなかった．しばらく息を潜めて待っていると，敵は簡単に姿を現した．これは敵は目が見えないに違いない．その後，部屋の中を逃げ回る敵を徹底的に追いつめる筆者は，悪戦苦闘の末になんとか捕獲に成功した．めでたし，めでたしである．

筆者の必死の格闘の様は，読者のご想像におまかせする．しかし，事務所では決して見かけなかったので，いないものだと思っていたのに……．いたことは，大きなショックだった．

それだけではない，不思議なことがある．事務所から自宅まで電源ユニットをもってくる長い間，なぜ，中からその虫が抜け出さなかったのか？

● その正体は……

その黒い虫とは，3億年前の石炭紀から生息し，最古の昆虫ともいわれるゴキブリだ．もっとも嫌われる害虫として，駆除の対象になっており，研究が進んでいる．ついでということで，この生態を勉強してみた．そうしたら後頭部を殴られるほどのショックを受けることになった．

成体のメスは交尾後，1年にわたって4回から8回に分けて産卵する．メスは，卵鞘（らんしょう）という名の卵の固まりを，体につけたまま移動するが，場合によっては唾液で卵鞘を，家具などの隅っこのほうに着けることもあるらしい．この卵鞘には，じつは30個以上もの卵が入っていて，3週間で一斉に孵化する．そして生まれた幼虫は約60日で成虫になる．

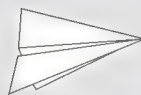
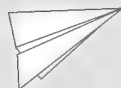
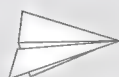
成虫の寿命は1年といわれているので，メスは一度の交尾で一生の間卵を生みつづけることになる．生まれた子供も，60日後には卵を産む．これから，試算すると，もし若い1匹のメスが侵入し，子供がすべて順調に成長したとすると，計算の上では1年後には，なんと1,000万匹を超えてしまうことになるという．すさまじいまでの繁殖能力である．

自分の糞のほか，木屑や塵，埃まで何でも食料とする．さらに，何もなくなると，共食いをしたり，卵鞘を食べたりするから，ほとんど食料に事欠くことはない．

とくに好きな場所は，湿ったところ，そして暗くて狭い場所だ．普段は集合して隠れ家を作って生活することが多い．とくに5mm以下程度の狭い幅を好み，入り込むようだ．糞には，集合フェロモンが含まれており，幼虫だけでなく，近くから成虫も寄せ集める威力がある．

夜行性で，昼間は隠れ家からまったく出ない．たまに，昼間に見かけるのは，増えすぎて餌が不足した場合などに，新しい天地を求めて移動する時だという

目は二つではない．トンボとまではいかないが数千の複眼を持っている．実際には物を影として捉えることはできても，人が近くにいても認識できない．ただし，人が動くと，そのときできたわずかな風を触覚で感知して気が付く（すごい！）．

そんな彼らにも弱点がある．一つは，水分がなくなると死ん

でしまうことだ．だから湿ったところを好み，体内の水分を維持するため体が脂ぎっている．しかし，1滴の水を与えただけで，最高3か月生きたという実験結果もある．水がなくても湿った埃や，壁紙やガラスにつく水分などでも十分に生きていける．

もう一つの弱点は，寒さだ．これが，北海道にはいない理由でもある．一般的には，冬には死滅するが，メスの死体に卵鞘としてくっつきながら極寒の時期を過ごし，春になると幼虫となって再び現れるのが普通だ．ただし，最近は暖房の普及もあって，冬場も生き延びて繁殖することが多いそうだ．

● 冬場にはパソコンを動かすな

冬場のオフィスは絶好の棲家でもある．暖房があって暖かいし，窓際や壁紙には十分な水分もある．お弁当や，お菓子などの残りカスが落ちていることも多い．これは絶好の条件だ．

電源ユニットに入っていたのは，電源内部が暗くて暖かかったこと，そして入り口が彼らにとってちょうどいい隙間に見えたことだろう．ほかにも，パソコンケースは前面の下側に，細い吸気口があるのが一般的だ．こんなところから，パソコンケースの中に入ってしまえば，中は結構暖かい．

その日，最後の社員がパソコンの電源を切り，オフィスの明かりを消して帰る．暗闇の中，彼らはパソコンからおもむろに出てきて，餌を探す……といった光景が，じつは毎夜繰り広げられているのかもしれない．

いやうちは大丈夫．オフィスに彼らはいない．そう思っても，安心できない．彼らは，餌が少なくなる冬場には，歩いてやってくる．それだけではない，オフィス家具やパソコンを運び込むと，それに付いてくる可能性もあるのだ．

筆者が以前勤めていた会社ではソフト開発を行っていた．事務所は新宿の高層ビルだったが，徹夜の作業中によくこの種の害虫を見かけたものである．当時は，高層階のレストラン街から，荷物運搬用エレベータで降りてきたのだろうと思っていたが，もしかしたらそうではなかったのかもしれない．なぜなら，お客様用のパソコンを使って開発し，開発が完了すると納品するということもやっていたからだ．また，トラブルが発生するとお客のパソコンを一時的に借りてきて，作業を行うということは日常茶飯事だった．

この会社では，連休などを使って定期的に害虫駆除を業者に

依頼していた．業者は，局所的に薬剤の注入や塗布をするほかに，殺虫剤の空中散布もするらしい．神経質な筆者が，「接触不良などを起こす可能性があるパソコンにかからないようにして欲しい」と要望したら，「大丈夫です．パソコンなどには，最初からビニール布を被せて散布しますから」といわれた記憶がある．今でも同様ならば，業者に依頼してもパソコンの中の彼らは死滅する可能性が低いわけだ．しかも，卵鞘に残っている卵は，殺虫剤では死滅しない．したがって，ここから幼虫がでてくる3週間後に，再度駆除しないと，本当は効果がないということにもなる．

とりあえず，とくに冬の時期，中にメスや卵鞘が入っているパソコンをオフィスで出し入れしたりすれば，春にはたいへんな目にあうかもしれないということを，肝に銘じておくべきだろう．

先の電源ユニットは，ビニール袋に何重にも厳重に包まれベランダに放り出してある．寒空の下，ベランダで春が来るのを待っている．

あさひ・しょうすけ　テクニカルライター
イラスト　森　祐子

Engineering Life in SiliconValley

目に見えないシリコンバレーの成功要因

H. Tony Chin

筆者は，こちらでさまざまな日本人と会う機会がある．多くは，エンジニアやテクノロジ関係の企業で働いている人，エンジニアリングに深く関係のある人(たとえばベンチャーキャピタルや工学部の教授など)だ．いろいろな話の中で，こちらの景気や日本での景気が話題になるが，やはり多いのは「なぜ日本のエンジニアは起業しないのか？」ということだ．日米でそれぞれ勤務経験のあるエンジニア達は，一致して「日本のエンジニアはレベルが高く，純粋なスキルではシリコンバレーエンジニアと見劣りしない」という．そしてここから一歩進み，「それではなぜ日本でハイテクベンチャーがあまり育たないのか？」という議論になり，さまざまな意見が出ることになる．

☆ ビジネスセンスを重視した地域

このような議論では，まずシリコンバレーの環境的な良さがあげられる．たしかに，UC Berkeley や Stanford など，工学部や科学が強い大学が近くにある．起業の種になりそうなアイデアや人材も密集しているかもしれない．また，まわりにスポンサーになってくれるベンチャーキャピタルもいる．このように環境が優れているということで，アメリカ国内や国外でも，多くの場所で似たような要素を取り入れようという試みがあった．いずれも大学を中心とした産学一体となった地域作りだ．日本でも，大学を中心にしたシリコンバレーライクな街作りの例はいくつかあると思う．

しかし，シリコンバレー以外の場所での試みは，シリコンバレーを越えるほどの大きな成果がまだ出ていない．細かいところでは法律の整備，たとえばアメリカ並みのストックオプションの配布であるとか，ベンチャーキャピタルやエンゼル投資家がキャピタルロスを税金控除額として使えることなど，いろいろな違いがある．筆者の見解では，シリコンバレーには相応のプロが集まりやすく，ビジネスセンスのある人の密度が高いという点を感じる．これは，とくにエンジニアでもただ単にエンジニアリングが優れているのでなく，ビジネスセンスがかなりあると感じられる．もちろん他のプロ，たとえばベンチャーキャピタル，知的所有権の専門弁護士，人事コンサルタント……すべてテクノロジ関連のビジネスにかなりの知識と理解が豊富だし，ビジネスセンスも長けている人たちが多い．

☆ ビジネス的知識を常にレベルアップさせる

筆者は，さまざまな国のエンジニアと仕事をする機会があったが，シリコンバレーのエンジニアにはたしかに独特の雰囲気がある．とにかく情報収集が好きだし，話題や情報も豊富にもっている人が多い．これはやはりアメリカの職場の環境の特徴なのだろうか．自分のストックオプションや401K個人年金の運営が個人の責任なので，あまりビジネスに興味がなくても，自分の会社の業績や株式市場に敏感になることが多いからであろう．年金や投資信託，そして自分の会社の持ち株会(ESSP：Employee Stock Purchase Plan)など，あらゆる場面で株式市場の知識が必要とされる．少なくとも自分の貯金や財務運営などには必須なので，嫌でも少しずつ身に付けていくようだ．

それにとどまらず，本格的に興味をもつエンジニア達もまた多い．そして会社の業績を発表しているプレスリリースにしっかりと目を通したり，それによる業界紙やウォール街の反応を細かくチェックすることが当たり前となる．アニュアルレポートの内の財務報告書もしっかりと読めて，経理・財務に詳しいエンジニア達も多い．社外の第三者的存在からの会社の評価や，将来性の情報を集めているわけだ．

また，たいていのシリコンバレーの企業では，社員にオーナー的精神と運命共同体であるという意識を感じてもらうために，社内に情報をかなりオープンにしていることが多い．社内向けの業績説明会やデータの発表が頻繁に行われる．かなりオープンな会社であれば，この種の情報で自分の仕事と業績がどのように直結しているか，かなり理解を深めることができる．

以前に筆者の勤めたことのある会社では，だいたい四半期ごとに事業部全員を集めて説明会があった．朝食まで用意してくれて，情報開示以外にその四半期の各部署の社員表彰式の場などに使われており，2～3時間は使ったけっこうなイベントだった．社員全員が株主なので，ミニ株主総会のような感じで，上層部による質疑応対もていねいに行われる．最近聞くところによると，Web キャストなども使われているようだ．これによって，社内と社外でピックアップできる自社の評価でエンジニア達は自分の勤めている会社について判断することが多い．しかし，矛盾があったり，社外の評価があまりよくないと，やはり転職を考えるという方向に考えが向いていくようだ．会社を辞める/辞めないはさておいて，ビジネス的な基本知識やベースとなる決算書の解読方法などを確実に身に付けられるのだ．

☆ 風土的な要素

まわりの転職のペースも速いので，異なったバックグラウンドの人と仕事をしたり接したりする機会も非常に多い．これは何度もこのコラムでも書いたが，自分のコネやネットワークを作るという点に関しては，非常にパフォーマンスが高い．逆にヨーロ

ッパのように法的なしがらみで，なかなかすぐに辞めたり転職できない環境であると，一つの会社に留まることになる．シリコンバレー以外のアメリカ国内でもそうだ．シリコンバレーは西海岸であるが，東に行けば行くほどなかなかエンジニアは転職しなくなる．これは筆者以外にも感じている人が多い．はっきりとした理由は筆者もよくわからないが，東部では企業の城下町のような都市がまだ多いし，保守的な風土がある．だから日曜日にはちゃんと教会や礼拝にいったり，日曜日には町中のすべてのお店が閉まるという町は，まだ東部のほうでは当たり前だ．長時間勤務したり家から仕事をする場合も多いが，東部では家族中心の生活などもまだまだ多い．シリコンバレーでは，スポーツカーを運転した独身成金が多い．これは東部ではなかなか見られないと思う．

東部のエンジニアは，一社で勤め上げるとまではいかなくても，しっかりと結果を出してから辞めるというケースが多いと思う．地道に積み上げたスタイルのエンジニア達が多いし，平均的に優秀なエンジニア達が多い．一方のシリコンバレーでは，きっかけやチャンスを重んじて勝負に出るケースが多いようだ．だから引き際も早く，前述の社内外の評価をベースに他のチャンスを求めるという風土がある．まあ，ただ単に腰が軽いともいえるが，扱っている情報量で見ると，同じアメリカでもシリコンバレーは過剰かもしれない．

結果として，情報収集力，コネ作りやきっかけ作りのパフォーマンスは非常に高いと思う．しかし，大きなデメリットもあり，エンジニアのスキル面ではもともと賢い人は情報を消化していってどんどん賢くなるが，それほどできの良いエンジニアでない人は地道にやるしか純粋なエンジニアリングのスキルがレベルアップしないかもしれない．シリコンバレーではできの良いエンジニアは本当に凄いが，そこそこのエンジニアやあまりレベルの高くないエンジニアも多く，エンジニアの層が厚いといつもいわれている．早い転職のペースは一長一短というところだろうか？

☆ 元エンジニアが多い土地柄

では，ずっとエンジニアリングを続けないエンジニア達はどうなるのか？　多くの場合，他の職種に目覚めてその道をめざすケースが多い．シリコンバレー企業の営業マンやマーケティング関連のスタッフは「元エンジニア」が多い．シリコンバレーにある弁護事務所の弁護士の多くも元エンジニアが多い．純粋な技術系の仕事でキャリアを作り上げるよりも，文系の仕事と融合させたほうが自分の性格に合っていると判断した人達だ．悪い言い方をすれば，「エンジニアくずれ」かもしれない．しかし，元エンジニアで営業をしている人達などは，やはりベースになるエンジニアリングの知識があるので，そこでメリットを発揮する．複雑なシステムを扱ったり，またエンジニアが使うツールや部品を扱うに営業をする場合，当たり前にしっかりとしたエンジニアとしての知識が必要になる．これによって，以前エンジニアをやっていた営業スタッフを抱えることが必須となる．知的所有権専門の弁護士なども，エンジニアリングの経験が必要な仕事だ．

そのほかに，広報担当の会社のスタッフや人事派遣・斡旋の会社でも元エンジニアが多い．いずれも顧客がかなり技術的に複雑な製品を扱ったりするからだ．たとえば人事派遣・斡旋会社でも，クライアント企業の技術と業界を熟知している元エンジニアが担当するケースが多い．ベースとなるエンジニアの知識が高いので，クライアント企業もあまり細かいことを言わずに済む場合が多い．これらの元エンジニアが多数であることもシリコンバレー独特なところではないだろうか？　他の場所，アメリカ国内でも少ない傾向だと思う．

☆ 日本のエンジニアもビジネスセンスを！

総合して考えると，たしかにエンジニアリングや物作りが三度の飯よりも好きな根っからの技術者も，シリコンバレーにはたくさんいる．そして彼らが考え出すアイデアなどが新しい企業へとつながっている．しかし，エンジニアとしてのスキルとそのプラスアルファ的な要素が，じつはもっとたいせつだと感じる．つまり，エンジニア以外でのスキル，とくにエンジニアリングをビジネスにつなげるという知識レベルが高く，また共通知識として普及していると思う．日本のエンジニアリングのレベルは非常に標準が高いと思うので，このプラスアルファのところを何とかすれば，日本でもシリコンバレーに負けないハイテク企業が育っていくのではないだろうか．

トニー・チン　htchin@attglobal.net　WinHawk Consulting

Column

シーズからのベンチャー

日本発では，シーズをベースにしたベンチャーが良いとされるケースが多い．つまり大学で，あるアルゴリズムをベースに製品化をしてビジネス展開をしていこうというアイデアだ．目新しいアルゴリズムだと，たしかに学界での評価は高いかもしれない．しかし，商品化となるとただ作ればよいというものでもないと思う．たとえば，優れたシミュレータのアルゴリズム（エンジン）があったとしても，既存のデータを取り込む部分の作り込みが必要となる．また，アルゴリズムを実験するプロトタイプができたとしても，実際の顧客のデータを使った設計フローで実証されなければ使い物にならない．意外と外部からデータを取り込むインターフェースやデータベースの作り込みの部分で時間と労力を使うケースが多い．

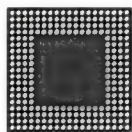
また，最終製品を取り巻く市場状況も成功を左右させる．たとえば，どんなに優れた新しいプロセッサを開発したとしても，それだけでなくファームやソフトを開発する環境を用意したりするのは当たり前だし，どの市場にまず売り込むかによっても大きく戦略は変わる．これからプロセッサを売り込むといっても，ウィンテル系が市場を制圧している PC 向けに売り込んでいくのは非常に難しい．エンジニアとしてシーズでビジネスを展開していきたい気持ちは大きいと思うが，肝心なのは，そのシーズを取り巻く環境をどうおさえていくかであろう．

HARD WARE

●システムインパッケージ

HJ931201BP

- SuperHマイコンである「SH-3」CPUコア製品をベースに，64MビットSDRAMと16Mビットフラッシュメモリをワンパッケージ化．
- 3段スタック構造の採用により，従来各パッケージで構成した場合と比較して，大幅に実装面積を削減でき，システムの小型化が図れる．
- マイコンとメモリの混載により，ユーザーによる両チップ間のバス設計が不要となり，短期間での製品開発が可能となる．
- 小型化により基板配線が短くなったため，EMIノイズの影響が減少し，安定した高速動作を実現可能．

■（株）日立製作所
サンプル価格：¥5,000
TEL：03-5201-5083
URL：http://www.hitachisemiconductor.com/

●ネットワークプロセッサ

NS7520

- 32ビットRISCプロセッサ「ARM7」をコアに，10/100Base-T Ethernet MAC，13チャネルDMAコントローラ，メモリコントローラなどの主要周辺回路を1チップ化．
- 動作周波数32～55MHz，177ピンBGA（13×13mm）のパッケージで提供．
- 10/100MビットMIIベースのPHYインターフェース，512バイトの送信および2Kバイトの受信バッファを内蔵．
- Ethernet送受信×2，シリアル送受信×4，外部DMAポート対応×4，メモリtoメモリ×3のコントローラを搭載．
- 2チャネルシリアルポート（HDLC，UART，SPI）をサポート．

■ネットシリコン ジャパン（株）
価格：$7.95～$9.95（10,000個時）
TEL：03-5428-0261　FAX：03-5428-0262
URL：http://www.netsilicon.co.jp/

●16ビット1チップマイコン

H8S/2628F

- CANインターフェース内蔵マイコン．
- シリアルコミュニケーションインターフェース（SCI）に加え，新たなモジュールとして，チップセレクト付き高速同期式シリアルコミュニケーションインターフェースを搭載．
- 8，16，32ビットの連続転送が可能であり，最大6Mbpsまでの転送速度を実現．
- EEPROMやセンサなどの外部デバイスとの高速通信が可能．
- Bosch CAN Ver2.0B active規格に準拠したHCANを内蔵．
- CANインターフェースはデータバッファを16メッセージ格納可能で，最大通信速度1Mbpsを実現．
- 低消費電流モードであるスタンバイモードからの復帰用に，CANのバス動作でマイコンをウェイクアップする機能を内蔵．
- CANインターフェースを搭載した16ビット1チップマイコンでは，最高速である24MHzの最大動作周波数を実現．

■（株）日立製作所
サンプル価格：¥1,700
TEL：03-5201-5212
URL：http://www.hitachisemiconductor.com/jp/

●MOSFETシリーズ

IRFPS35N50L/IRFP31N50L/IRFP23N50L/IRFB17N50L/IRFIB5N50L

- 耐圧500VのHEXFETパワーMOSFETで，高速のファストリカバリダイオードを内蔵．
- 通信機器などで使うZVS方式のスイッチング電源向けに最適化．
- ダイオードを新たに外付けする必要がなく，部品点数を削減でき，プリント基板への部品の配置が簡略化できる．
- 内蔵ダイオードの逆回復時間は250ns以下で，従来品と比較して70%高速化．
- 逆回復電荷を70%以上減らしているため，スイッチングの損失も低減．

■インターナショナル
レクティファイアー ジャパン（株）
サンプル価格：¥600～
TEL：03-3983-0875　FAX：03-3983-0642

●クロックマネジメント

TeraClockファミリ

- ゼロディレイバッファとプログラマブルスキュー製品からなる，クロックマネジメント製品群．
- 入力用5本，出力用4本のユーザー設定可能なシングルエンド信号または差動信号を備えた，さまざまなI/O間での変換をサポート．
- 製品間での相互移行を容易にするピン互換性をもつ．
- 高速通信アプリケーションでの使用に適する多くの機能を提供．
- PLL機能による信頼度の高い信号を提供し，さらに冗長クロックとヒットレススイッチオーバ機能を活用して，システムのメインクロック信号が中断された場合にはシステム全体の完全性と信頼性を維持，確保できる．
- サイクルごとのジッタが50psまた出力スキューが100psでありながら，最大250MHzの周波数で動作する．

■日本IDT（株）
サンプル価格：$4.50～$16.25（10,000個時）
TEL：03-3221-6726　FAX：03-3221-5456

●RFシングルチップ

ZL20200

- 送受信回路を一つのシリコンに集積したデバイスで，携帯電話の無線部分のサイズとコストを大幅に削減することが可能．
- 100dBの利得制御送信と，AMPS（Advanced Mobile Phone System）FM復調の処理が可能．
- ベースバンドフィルタを組み込んでいるため，外付け回路を使うことなく，ほとんどの市販ベースバンドプロセッサに直接接続できるようになる．
- トランスミッタは，直角変調器，IF利得制御器，シングルエンド出力器を装備したリニア回路．
- シングルエンド出力器は，2ステージパワーアンプを直接駆動するのに十分な電力の供給が可能なため，外付けのバラントランスが不要．
- フラクショナル-NシンセサイザDを利用してUHF VCOを制御．
- TDMAモードとAMPSモードでは850MHzの携帯電話周波数で作動し，PCSバンドのTDMAモードでは1900MHzで動作．

■ザーリンク・セミコンダクター・ジャパン（株）
価格：$5以下
TEL：03-5733-8181　FAX：03-5401-2441

●DDRターミネーションレギュレータ

LP2996

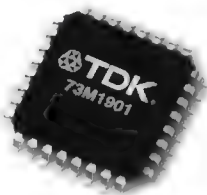
- SSTL-2仕様に準拠しているJEDEC DDRバスターミネーションレギュレータ．
- スイッチングノイズの除去，シグナルインテグリティの改善，最大60%のシステムコストの低減を実現．
- シンクおよびソースを連続電流で1.5Aに抑えることが可能で，オンチップの過熱防止機能，シャットダウンピン，チップセットおよびDIMMS向けのリファレンス出力，負荷調整を向上させるセンスピンを装備．
- 電源およびアナロググレール向けに独立した入力を装備．
- 電力回路の電圧を下げることができ，1.8Vレールからの変換によって内部のパワー損失を低減する．
- 外部リファレンス入力の半分のアウトプットを生成する内部レジスタディバイダが組み込まれている．
- SSTL-2，SSTL-3，HSTLといった仕様向けにアクティブターミネーション電圧を生成できるように拡張可能．

■ ナショナル セミコンダクター ジャパン(株)
価格：¥115（1,000個時）
URL：http://www.national.com/JPN/

●モデム用アナログフロントエンドIC

73M1901

- 16ビットのCODECを搭載し，2400bps（V.22bis）から56Kbps（V.92）までをサポートするソフトモデム用アナログフロントエンドIC．
- 差動ハイブリッド回路を採用したことによって小信号時の*SNR*を改善．
- 従来の標準的なトランスタイプのDAAとの互換性を確保し，二つの低消費電力モードと七つのテストモードを装備．
- JATE，CTR21，FCCその他の世界中の通信規格に合わせて設定することが可能．

■ TDK（株）
サンプル価格：¥1,500（10個時）
TEL：03-5201-7231

●CANトランシーバ

SN65HVD251

- 5V動作のCANトランシーバ．
- 「PCA82C250」および「PCA82C251」とピン互換．
- ±36Vのバス端子耐圧を実現．
- 12kV（HBM）を超えるバス端子ESD耐圧．
- ISO 11898規格に準拠．
- 最大1Mbpsの差動信号伝送をサポート．
- 高い入力インピーダンスにより，バス上に最大120個までのノードを接続可能．
- 電源オフのノードによるバスへの悪影響を防止．
- 低電流スタンバイモードをサポート．
- サーマルシャットダウン機能搭載．
- 電源オン/オフ時のグリッジ発生防止機能により，活線挿抜に対応．

■ 日本テキサス・インスツルメンツ（株）
価格：¥125（1,000個時）
TEL：03-4331-2740　FAX：03-4331-3205
URL：http://www.tij.co.jp/analog/

●プラグ＆プレイBluetoothモジュール

BGB201 TrueBlue 半導体モジュール

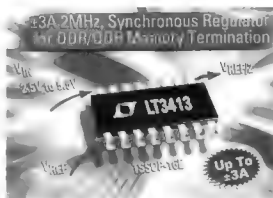
- 1個の統合モジュールにベースバンド機能とラジオ機能の両方を搭載．
- ベースバンド部には，224Kバイトの組み込みフラッシュメモリ付きARM7マイクロコントローラ，およびUSB，UART，PCM，GP I/Oなど多様なインターフェースを1個のダイに搭載．
- 既存のBluetoothデータおよびボイスパケットすべてに対応し，モジュールを実装する機器に備えてあるボイスCODECに接続して音声機能を実行することも可能．
- ラジオ部には，ニアゼロIFトランシーバチップ，バラン回路，Tx/Rxスイッチ，バンドパスフィルタなど重要なRF部品を搭載．
- 必要な外付け部品は，外部クロックソースとアンテナのみ．

■ ロイヤルフィリップスエレクトロニクス社
価格：下記へ問い合わせ
URL：http://www.philips.co.jp/semicon/

●同期整流式降圧レギュレータ

LTC3413

- DDR/DDR2メモリアプリケーション向けにバス終端電圧を生成可能な高効率なモノリシック同期整流式降圧スイッチングレギュレータ．
- 出力電流は最大3A．
- 2MHzという高いスイッチング周波数が可能なため，小型の外付け部品を使用可能で，実装面積の削減が可能．
- RDS（ON）がわずか85mΩの内蔵スイッチにより，90%の高効率を達成すると同時に，0.6Vの低出力電圧を生成．
- 熱特性が改善されたTSSOP-16Eパッケージで供給されるので，耐熱性に優れる．
- 2.25V～5.5Vの入力電圧範囲で動作し，VREF/2に等しい安定化出力電流を供給する固定周波数電流モードアーキテクチャを採用．

■ リニアテクノロジー（株）
サンプル価格：¥485（1,000個時）
TEL：03-5226-7291　FAX：03-5226-0268

●LVDS出力SAW発振器

EG-2121CA-L（2.5V）
EG-2102CA-L（3.3V）

- LVDS出力のSAW発振器．
- 出力仕様をLVDS出力にすることにより，低消費電力，周辺回路の簡素化，ノイズの低減などが可能となり，FPGA，ASICなどの回路入力インターフェースに適した仕様となっている．
- 安定したデューティ比（±2%）によりDDR RAM用途やシリアル通信（Serial-ATA，Fibre Channel，PCI-Express，Ethernet，Hypertransport，IEEE1394bなど）を内蔵した機器の構築が可能．
- 外形サイズ5×7×1.4mmのセラミックパッケージに高精度SAW共振子および新開発のICを搭載．

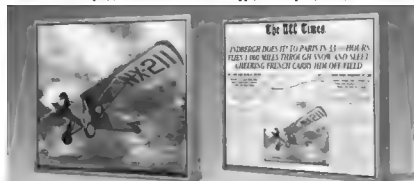
■ セイコーエプソン（株）
価格：下記へ問い合わせ
TEL：042-587-5878
E-mail：ED_QD_Marketing@exc.epson.co.jp

HARD WARE

●液晶ディスプレイモジュール

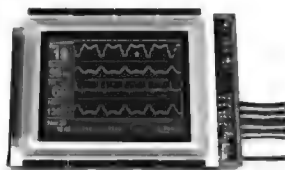
QSXGA カラー LCD モジュール

- Quad-SXGA(2,560 × 2,048 ピクセル表示)の大表示容量を実現し，ブラウン管モニタでは困難であった CAD，CG や X 線写真の表示など，高精細，高解像度表示を必要とする用途に適する．
- SA-SFT 技術の採用により，見る方向によって同色でも明るさが違って見える γ 変化や，色の反転が起こる色変化が少ない視野角特性を実現．
- カラーモデルは RGB 各 8 ビット入力，1677 万色表示に対応し，EBU 規格 100%対応の色再現範囲を実現．
- モノクロモデルは，各サブピクセル 256 階調表示により，1 ピクセルあたり 766 階調の表現力を実現．

■ 日本電気（株）
価格：下記へ問い合わせ
TEL：044-435-1851
URL：http://www.ic.nec.co.jp/compo/lcd/index.html

●ディスプレイモジュール

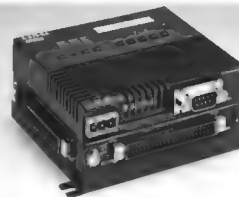
マルチカラー無機 EL ディスプレイモジュール

- カナダのアファイヤー・テクノロジー社との技術提携をベースに開発されたマルチカラー発光で，200cd/m² 輝度の小型カラーディスプレイモジュール．
- 厚膜工法による完全な固体電界発光という独自の技術により，耐振動性，耐ショック性，耐環境動作性でもすぐれた特性を示す．
- モジュール外形は 146 × 108 × 15mm で，表示部寸法は 86 × 65mm．
- ドット構成は 240 × 180 で，消費電力は白色時で 10W．
- 線順次駆動による駆動方式を採用．

■ TDK（株）
価格：下記へ問い合わせ
TEL：03-5201-7102

●ステッピング/サーボモータ用 2 軸コントローラー

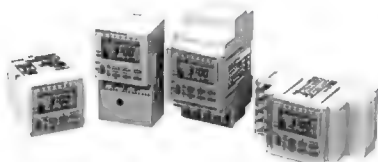
C-570-SA

- 発信器，表示パネル一体型で，従来品と比較して大幅に小型化．
- 設定された DRIVE，シーケンサの演算結果で DRIVE，センサを利用した DRIVE，回転制御で 360 °を最短で DRIVE する近回り DRIVE など，豊富な DRIVE が多様なモータ制御に対応．
- 専用ソフトウェア MAP-12-SWXP を使用して，パソコンからデータの設定/編集が行える．
- タッチパネルなどにより，上位シーケンサの I/O 信号を通して 50 ポイントの設定が可能．
- 軸の追加にも新たな設置スペースを設けることなく取り付けることが可能．
- パネルの 7 セグメント上にて各種状態をモニタすることが可能．

■（株）メレック
価格：下記へ問い合わせ
TEL：0426-64-5383

●タイムスイッチ

形 H5F デジタル・デイリータイムスイッチ

- 1 日 24 ステップ(12 セットのオン/オフ動作)のオン/オフ時刻制御の設定が可能．
- 動作曜日の選択も可能で，月～金曜はオン/オフ動作，土～日曜は出力オフさせるなどの用途に適する．
- CE マーキングに対応したほか，UL/CSA 規格も取得し，輸出装置などグローバルでの使用が可能．
- 設定したプログラム内容を確認できるテストモード機能，プログラムの変更無しに祝祭日などの休日に対応できる休日機能，ワンタッチでサマータイムに変更可能なサマータイム機能などを搭載．
- 取り付け方法は，埋込み取付け，表面取り付け，協約寸法サイズ（縦/横）の四つの取り付けタイプを用意．

■ オムロン（株）
価格：¥15,300　TEL：075-344-7119

●パルスデータ管理

Pulse Recorder RPR-72

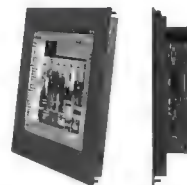
- パルス信号を測定，記録し，記録データを特小電力無線通信機能によってパソコンに送り，データ処理ができる．
- データの自動収集，異常監視機能などの多彩な機能をもつ．
- 測定は，記録間隔内のパルス数の記録，発生したパルスのイベント時刻の記録の 2 通りをサポート．
- パルス数を任意の単位にスケール変換して，本体液晶に表示することも可能．
- RS-232-C を備えた機器のデータの記録が可能．

■（株）ティアンドデイ
価格：¥42,800
TEL：0263-27-2131　FAX：0263-26-4281
E-mail：info@tandd.co.jp

●パネル型コンピュータ

パネルコンピュータ 630 シリーズ

- 産業向けタッチパネル機能付き表示器，CPU 一体型パソコン．
- Low Power Mobile Pentium Ⅲ 800MHz (FSB 133MHz) CPU を搭載．
- 自然空冷でファンレス運転を実現．
- 組み込み向け CPU，チップセットを採用し，長期安定供給を実現．
- 自社カスタマイズの Phoenix 社製の BIOS を採用し，BIOS レベルのサポートが可能．
- EEPROM による CMOS データの保持で，バッテリ切れ時も起動可能．
- PC カードスロット，100Base-TX LAN を標準装備．

■（株）コンテック
価格：¥405,000 ～ ¥480,000
TEL：03-5628-9286　FAX：03-5628-9344
E-mail：tsc@contec.co.jp

●産業用コンピュータ

ボックスコンピュータ 630シリーズ

- 衝撃，振動に高耐久性をもつファンレス稼動の産業向けパソコン．
- Low Power Mobile Pentium Ⅲ 800MHz (FSB 133MHz) CPUを搭載．
- 自然空冷でファンレス運転を実現．
- 組み込み向けCPU，チップセットを採用し，長期安定供給を実現．
- VGA変換アダプタ付きのDVIビデオ出力搭載．
- 自社カスタマイズのPhoenix社製のBIOSを採用し，BIOSレベルのサポートが可能．
- EEPROMによるCMOSデータの保持で，バッテリ切れ時も起動可能．

■（株）コンテック
価格：¥285,000～
TEL：03-5628-9286　FAX：03-5628-9344
E-mail：tsc@contec.co.jp

●ファンクションジェネレータ

SG-4115

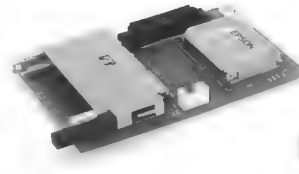
- ダイレクトディジタルシンセシス方式を採用し，15MHzの正弦波，方形波出力，三角波，ランプ波，直流，ノイズ波形出力のほか，任意波形（ARBキー使用）も2チャネル独立に高精度，高品位で出力できる．
- 基本操作はダイレクト操作で設定可能．
- 蛍光表示管に電圧と周波数を同時表示でき，表示切換えで9桁の周波数設定も可能．
- 2チャネルを独立発振させることも位相を合わせることも可能．
- 任意の波形を合計4本まで内部に保存可能で，パソコン上で波形を作成可能なユーティリティソフトウェアを用意．
- 専用ASICによるSWEEPを行っているため，なめらかなSWEEPが可能．

■岩通計測（株）
価格：¥348,000
TEL：03-5370-5474　FAX：03-5370-5492
E-mail：info-tme@iwatsu.co.jp

●インテリジェントGPS

iiGPS

- 名刺大のスペースに，LinuxPC，高精度GPSおよび通信モジュールの3ユニットを統合化したインテリジェントGPS．
- GPSモジュールは米国トリンブル社のFirst Technologyを用いており，単独測位時に7m，RTK-DGSP補正を行い1mの測位精度を確保．
- VRS-DGPSデータは，国土地理院が設置した電子基準点よりRTK用補正データをもとに生成されたVRSデータの一部を使用．
- 消費電力の小さいGPSモジュールと，RISC CPUを組み合わせて使用し，内部にパワーマネジメントを組み込むことで，省電力を実現．

■（株）ジーシーシー
サンプル価格：¥30,000
TEL：03-5209-6960

●形熱画像計測装置

CPA-8000

- 形熱画像計測装置．
- 画像データ保存には内蔵RAM（最大900画像）およびコンパクトフラッシュカード（128Mバイト/最大900画像）の二つを装備．
- 画像更新周期は，60Hzのリアルタイム．
- 4形カラー液晶モニタは，本体から取り外して，リモートコントロールが可能．
- JPEGフォーマットでの画像保存が可能で，Wordでの解析，レポートの作成も可能．
- 画像読み出しソフトウェアを使用し，本体からパソコンへ画像を直接送信可能．
- 測定温度は，－40～120℃および0～500℃で，オプションにより2000℃まで対応．
- 非冷却マイクロボロメータ素子を採用．
- 7.5～13μmの長波長を採用．
- 精度は，測定値の±2%または±2℃の高精度で測定．

■（株）チノー
価格：¥4,280,000
TEL：03-3956-2449
FAX：03-3956-0459

●テストソフトウェア

ラボ・アプリケーション プロトコル・アプリケーション テスト・アプリケーション

- ラボ・アプリケーションは，RFパラメトリックテスト機能とプロトコルテスト機能の両方に対応．「Agilent E5515C ワイヤレス・コミュニケーション・テスト・セット」のRF入出力端子に携帯電話を接続し，LAN端子からPCや実網に接続することによって，Agilent E5515Cを実際の基地局に見立てた評価系の構築が可能．
- プロトコル・アプリケーションは，プロトコルテスト機能のみに対応し，携帯電話のハードウェアとその上で動作するソフトウェアを統合する際の評価やファンクションテストで使用．
- テスト・アプリケーションは，RFパラメトリックテスト機能のみに対応．

■アジレント・テクノロジー（株）
価格：約¥1,500,000～¥4,100,000（ラボ・アプリケーション）
約¥1,800,000（プロトコル・アプリケーション）
約¥1,400,000～¥1,700,000（テスト・アプリケーション）
TEL：0120-421-345

●Ethernetスイッチ

CentreCOM 9800シリーズ

- 32Gbpsのスイッチングファブリックを備え，ワイヤスピード/ノンブロッキングのレイヤ2/3スイッチングは，スタティックルーティング，RIPv1/v2，OSPFのダイナミックルーティング，およびDVMRPのマルチキャストルーティングに対応．
- ハードウェアによるパケットフィルタを装備．
- IP/UDP/TCPヘッダ内のパラメータ，プロトコル，MACアドレス，VLAN，ポートなどの指定ができ，パケットの破棄，転送処理をワイヤスピードで実現することが可能．
- IEEE802.1D準拠のスパニングツリープロトコルにより，ネットワークの二重構成が可能．
- 帯域制御，優先制御，輻輳制御などのQoS処理をポリシーベースで実現可能．ポリシーは複数のトラフィッククラスで構成され，それぞれのクラスタに応じたサービス品質を定義可能．

■アライドテレシス（株）
価格：¥798,000（9812T）
¥698,000（9816GB）
TEL：0120-860-442
URL：http://www.allied-telesis.co.jp/

HARD WARE

●半導体ディスク

QikDATA M3

- Qikシリーズの半導体ディスク．
- 既存のシステムに接続し，システム内に点在するホットファイルを移動させることで，パフォーマンスの向上と磁気ディスクの数百倍のレスポンスタイムを実現させる，SAN対応のアクセラレータ．
- 50,000IOPS以上のデータ処理能力で，アクセスタイムは25μsの高速データ処理を実現．
- ミラーリングされた2基のバックアップ用磁気ディスクと二重化された内蔵UPSで，データの保護機能を実現．
- ワンセットにつき，同製品を最大7台（120Gバイト）まで接続可能．
- 前面LEDにより監視情報を常時表示する，セルフモニタリングを搭載．
- サイズは43.6×482×415mmで，重さは7.5kg．
- 対応OSは，WindowsNT/2000/XP，Linux，AIX，Solaris，HP-UXなど．

■（株）日立システムアンドサービス
価格：オープン価格
TEL：03-3763-1183　FAX：03-3763-5536
E-mail：qik@hitachi-system.co.jp

● VoIPメディアゲートウェイ

SHOUT900

- H.323-SIPのプロトコルコンバージョンを行うSHOUTlinkの機能を包含し，単位面積あたり最大の回線収容密度と省スペースを実現．
- メディアゲートウェイ，ゲートキーパー，IP交換機能，SS7シグナリング，コールマネージャ，IVRおよびSIP-H.323のプロトコル変換機能，SIP-H.323でのB2BUA機能，IP/TDM機能をすべて含むオールインワン型の装置．
- PCIベースの筐体およびカード類により，1Uのサイズで最大8T1/E1（240チャネル）を収容可能で，約60%のコストパフォーマンスを向上．

■ net.com Japan Inc.
価格：¥3,000,000～
TEL：03-5749-7157　FAX：03-3243-9503
E-mail：sales@promina.co.jp

● IPアクセスルータ

GeoStream Si-R870

- インターネットVPNで，IPsecを用いてデータを暗号化する際に，最大800Mbpsの高速スループットを実現．
- ハードウェアによる24Mppsの高速ルーティング性能を保持しているため，高品質な通信を実現するQoS制御を行った場合でも，速度を低下させることなくルーティングが可能．
- 基本制御部（制御モジュール，スイッチモジュール）と電源の二重化のほか，ルータ本体を複数台設置し，ひとつのルータに障害が発生した際には，別のルータに自動的に切り替わるホットスタンバイ構成が可能．

■ 富士通（株）
価格：¥6,950,000～
TEL：03-3548-3648
E-mail：telecom@fujitsu.com

●ネットワークメディアプレーヤ

Play@TV

- （株）ソーテックとのコラボレーションによる，パソコンに保存してある動画，静止画像，音楽ファイルをネットワーク経由でTVに表示，再生を可能とするネットワーク対応のメディアプレーヤ．
- 有線LAN（10Base-T）および無線LAN（IEEE802.11b）の両方に対応．
- Windows Media Playerで再生できる動画ファイルに対応（DVD-Video/Video-CDを除く）．
- サムネイル表示で，登録ファイルをリモコンで選択可能．
- ジャンルごとや，アーティストごとにリストの作成が可能．
- AV機器への映像出力部にはS-Video端子やコンポジット端子のほか，高画質なD1端子など，多彩な出力端子を搭載．
- 音声出力もアナログステレオだけでなく，ディジタル音声出力を装備．

■（株）メルコ
価格：¥29,500
TEL：03-5326-3754

● IEEE802.11g準拠無線LAN製品

WLM2-G54/WLA-G54 WLI-CB-G54

- WLM2-G54は，2.4GHz 11Mbps無線LANインテリジェントアクセスポイントの基本機能を踏襲し，IEEE802.1x/EAP，SNMP，SpanningTreeなどのセキュリティ，管理機能を搭載．60℃までの耐環境性能により，苛酷な使用環境に対応し，別売のPoEを使用することでLANケーブルを通して電源の供給が可能．
- コアCPUにはモトローラの「MPC8241 200MHz」を搭載．
- WLA-G54は，IEEE802.11gに準拠したブリッジモデル．11Mbps無線LANブリッジモデルの基本機能を踏襲しながら，台数無制限のリピータ機能を追加．4ポートの10M/100MスイッチングHubを搭載．WAN/LAN全ポートがAUTO-MDIXに対応．PoEアダプタを使用して，LANケーブル経由の電源供給，外部アンテナ装着も可能．
- WLI-CB-G54は，2.4GHz/54Mbps対応CardBusスロット用無線LANカード．

■（株）メルコ
価格：¥128,000（WLM2-G54）
¥30,500（WLA-G54）
¥13,200（WLI-CB-G54）
TEL：03-5326-3754

● DivX再生対応DVDプレーヤ

KiSS DVDプレーヤ DP-450

- デンマークのKiSS Technology社が開発した，PCを経由することなく再生が可能な，DivX Videoの再生用DVDプレーヤ．
- 米国Sigma Designs社のプログレッシブDVDデコーダチップ「EM8500」を搭載．
- DVD，VideoCD，MP3にくわえ，DivX VideoをDVD-R/RW，CD-R/RWから再生可能．
- DVD-R/RW，CD-R/RWに保存したMPEG-1/2/4データの再生が可能．
- DVD-R/RW，CD-R/RWに保存したディジタルカメラなどのJPEGデータの再生が可能．
- ハイビジョン対応テレビにも，コンポーネント出力することが可能．
- オプティカル端子，S/PDIF端子により，5.1chのディジタルサラウンド出力が可能．

■（株）バーテックスリンク
価格：オープン価格
TEL：03-5259-5159

SOFT WARE

●クロスツールレンタル

クロスツールレンタル

- 開発のピーク時や，協力会社への発注，試用など，突発的にツールが必要となった場合に利用が可能．
- 必要な期間で，必要なライセンス数を自由に設定してレンタル可能．
- レンタル商品は，C/C++コンパイラ（XCC-V），アセンブラ（XASS-V），シミュレートデバッガ（XDEB-V），総合開発環境フレームワークを含む．
- オーダーは，同社サイトレンタル専用オーダーフォームから可能．
- 見積りなどの手間をかけずに，すぐ開発が可能．
- 対応マイコンは，M32R／M16C／V850／ARM／SH／H8S／FR／MC68xxx．

■ **ガイオ・テクノロジー（株）**
価格：¥20,000（月額）
E-mail：sales@gaio.co.jp
URL：http://www.gaio.co.jp/

●HAクラスタリングソフトウェア

LifeKeeper for Linux v4.2.0

- ハイアベイラビリティクラスタソフトウェアのバージョンアップ版．
- Red Hat Linux Advanced Server2.1とSuSE SLES 8のディストリビューションに対応し，RedHat7.1/7.2向けカーネルとして2.4.9-34をサポート．
- オプションは，I/Oフェンシングメカニズムとして STONITH をサポート．
- リソース管理オプションは，複数のリソースを起動/停止する機能を提供．
- ディスプレイオプションは，サービスとリソースの表示においてカラム幅と行の高さをカスタマイズすることが可能．
- Apache2.0をサポート．
- SAP Recovery Kitでは，SAP R/34.6をサポート．
- Samba v2.2.4以降のSambaに対応し，Samba構成ファイルの同期機能をサポートし，操作性を向上．
- データ破壊を防止するための保護機能の強化，マルチミラー機能の操作性の向上などのローカルディスク障害時の処理機能を提供．

■ **（株）テンアートニ**
価格：¥480,000
TEL：03-5298-2929　FAX：03-5298-2854
E-mail：LK-sales@10art-ni.co.jp

●セキュリティツール

ファイルロッカー

- eTRONカードを用いて，PCのファイルを暗号化するセキュリティツール．
- 高セキュリティで名刺サイズの非接触ICカードとPC本体にUSBで接続するICカードリーダで構成．
- ドラッグ＆ドロップの操作だけで，簡単にファイルの暗号化/複合化が可能．
- ファイル単位とフォルダ単位での暗号化/複合化をサポート．
- グループ機能により，ファイルにアクセス可能なユーザーをきめ細かく制御．
- 暗号化/複合化のための鍵は，eTRONカードの中に暗号化して格納されるため，パスワードなどを覚える必要がない．

■ **パーソナルメディア（株）**
価格：¥98,000
TEL：03-5702-7858
E-mail：sales@personal-media.co.jp
URL：http://www.personal-media.co.jp/

●組み込み機器用ソフトウェア

MDL-TCP/IP

- IPv4とIPv6の両方に対応したデュアルスタック構造のTCP/IPプロトコルスタックソフトウェア．
- IPv4，IPv6のそれぞれ単独での実装も可能．
- μITRON，Tornado/VxWorksといった組み込み用リアルタイムOS，あるいはIPSecやIKEを含む主要なRFCに標準で対応．
- オプション製品で，各種アプリケーションプロトコルにも対応．
- オプション製品の無線LANドライバと組み合わせることで，無線LAN環境の構築が可能．
- コード約80Kバイト，データ約30Kバイトのコンパクトなコードサイズを実現．
- 内部コピーを極力抑えた設計により，高速化を実現．
- マルチホーミングや組み込みプロトコルの構成が可能．

■ **（株）神戸製鋼所**
価格：下記へ問い合わせ
TEL：03-5739-6880　FAX：03-5739-6391
E-mail：vxe-sales@info.kobelco.co.jp

●開発ツール

Intel C++/Fortran Compiler 7.0

- 各種インテルベースのプロセッサに最適な実行ファイルの生成が可能．
- Intelコンパイラによってビルドされたアプリケーションは，プロセッサディスパッチ機能によりランタイムにプロセッサを識別し，そのプロセッサに最適化されたコードを実行．
- Itanium 2プロセッサのサポートなどの機能が追加された．
- IA-32 Microsoft Visual Studio .NET開発環境への統合．
- IA-32 Microsoft Visual C++，Compaq Visual Fortranとソース互換．
- 浮動小数点命令で優れたスループットを提供．
- データプリフェッチ機能を搭載．
- プロシージャ間の最適化および，プロファイルに基づく最適化を装備．

■ **エクセルソフト（株）**
価格：¥55,000
（Intel C++ Compiler 7.0 for Windows/Linux）
¥65,000
（Intel Fortran Compiler 7.0 for Windows）
¥89,000
（Intel Fortran Compiler 7.0 for Linux）
TEL：03-5440-7875　FAX：03-5440-7876
E-mail：intel@xlsoft.com

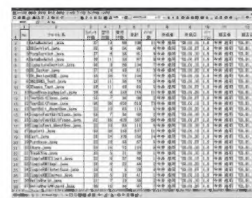
●ドキュメント自動生成ツール

A HotDocument for Visual C# .NET/for Visual J# .NET

- Visual Studio.NETの標準言語にすべて対応し，Access/Excel対応版のドキュメントも統一した形式での出力が可能．
- ソースファイルより20種類以上の高品質なドキュメントを自動生成．
- Visual Studio.NETの標準機能では得られないメソッド情報，インターフェース情報，コメント行，実行行の情報など納品物件に必要な情報を出力．
- 社内システムの内部資料，ソースファイルを解析するリバースエンジニアハングツールとして利用可能．
- 出力ファイル形式は，Excelファイル，テキストファイルの2形式をサポート．

■ **（株）ハローシステム**
価格：¥39,800
TEL：03-5367-5183　FAX：03-5367-5181
E-mail：info@hellosystem.co.jp

海外・国内イベント/セミナー情報

INFORMATION

海外イベント

1/27-30 **COMNET Conference & Expo**
Washington Convention Center, WA, USA
IDG
http://www.comnetexpo.com/comnetexpo/V33/index.cvn

2/4-6 **Digital Content Delivery Expo**
San Jose Convention Center, San Josa, CA, USA
PBI Media
http://www.replitech.com/

2/9-13 **International Solid-State Circuits Conference**
San Francisco Marriott Hotel, San Francisco, CA, USA
IEEE
http://www.isscc.org/isscc/

2/16-20 **Non-Volatile Semiconductor Memory Workshop**
Hyatt Regency Hotel, Monterey, CA, USA
IEEE
http://ewh.ieee.org/soc/eds/nvsmw/

2/18-21 **International Display Manufacturing Conference & FPD Expo**
Taipei Int'l Convention Center, Taipei, Taiwan
IDMC
http://osdlab.eic.nctu.edu.tw/idmc/

2/27-28 **Advanced Microelectronic Manufacturing 2003**
Santa Clara Convention Center and Westin Hotel, Santa Clara, CA, USA
SPIE
http://spie.org/conferences/programs/03/mm/

3/3-7 **Design Automation & Test in Europe**
Messe Munich, Munich, Germany
Europa Design and Automation Association
http://www.date-conference.com/

国内イベント

1/30-31 **Electronic Design and Solution Fair 2003**
パシフィコ横浜(神奈川県横浜市)
日本エレクトロニクスショー協会
http://www.edsfair.com/

2/5-7 **NET&COM 2003**
日本コンベンションセンター(幕張メッセ，千葉県千葉市)
日経BP社
http://expo.nikkeibp.co.jp/netcom/exhibit2003/

2/6-7 **第9回シンポジウム「エレクトロニクスにおけるマイクロ接合・実装技術」**
パシフィコ横浜会議センター(神奈川県横浜市)
(社)溶接学会
http://wwwsoc.nii.ac.jp/jws/research/micro/mate/Mate2003.html

2/12-14 **ISS Japan 2003 (Industry Strategy Symposium Japan)**
パンパシフィックホテル横浜(神奈川県横浜市)
SEMI
http://www.semi.org/web/japan/wexpositions.nsf/url/iss03_j

2/26-28 **IP.net JAPAN 2003**
東京国際展示場(東京ビッグサイト，東京都江東区)
リックテレコム
http://www.ric.co.jp/expo/ip2003/index.html

2/26-28 **国際ナノテクノロジー総合展・技術会議 nano tech 2003**
日本コンベンションセンター(幕張メッセ，千葉県千葉市)
nano tech実行委員会
http://www.ics-inc.co.jp/nanotech/

3/4-7 **IC CARD WORLD 2003**
東京国際展示場(東京ビッグサイト，東京都江東区)
日本経済新聞社
http://www.shopbiz.jp/2002/t_index.phtml?PID=0003&TCD=IC

開催日，イベント名，開催地，問い合わせ先の順

セミナー情報

DC-DCコンバータ設計の基礎
開催日時　：1月30日(木)
開催場所　：CQ出版セミナールーム
受講料　　：13,000円
問い合わせ先：エレクトロニクス・セミナー事務局，☎(03)5395-2125

オシロスコープ入門(1日コース)
開催日時　：1月30日(木)，1月31日(金)
開催場所　：日本テクトロニクス(東京都品川区)
受講料　　：20,000円
問い合わせ先：日本テクトロニクススクール窓口，☎(03)3448-3015
http://www.tektronix.co.jp/News/School/main.html

無線LAN構築・運用のトラブルシューティング
開催日時　：1月30日(木)～1月31日(金)
開催場所　：SRCセミナールーム(東京都高田馬場)
受講料　　：76,000円
問い合わせ先：(株)ソフト・リサーチ・センター，☎(03)5272-6071
http://www.src-j.com/seminar_no/23/23_019.htm

半導体製造装置入門
開催日時　：1月31日(金)
開催場所　：CQ出版セミナールーム
受講料　　：13,000円
問い合わせ先：エレクトロニクス・セミナー事務局，☎(03)5395-2125

C言語によるはじめてのLinuxプログラミング
開催日時　：2月5日(水)～2月6日(木)
開催場所　：オームビル(東京都千代田区)
受講料　　：92,000円
問い合わせ先：(株)トリケップス，☎(03)3294-2547，FAX(03)3293-5831
http://www.catnet.ne.jp/triceps/hi/linux.htm

リアルタイムOSの基礎
開催日時　：2月6日(木)
開催場所　：CQ出版セミナールーム
受講料　　：13,000円
問い合わせ先：エレクトロニクス・セミナー事務局，☎(03)5395-2125

ROM化C言語
開催日時　：2月6日(木)～2月7日(金)
開催場所　：江坂研修会館(大阪府吹田市)
受講料　　：20,000円
問い合わせ先：三菱電機セミコンダクタ・アプリケーション・エンジニアリング(株)半導体研修センター，☎(03)5783-7365
http://www.semicon.melco.co.jp/

Linuxデバイスドライバ開発入門
開催日時　：2月7日(金)
開催場所　：CQ出版セミナールーム
受講料　　：13,000円
問い合わせ先：エレクトロニクス・セミナー事務局，☎(03)5395-2125

オシロスコープ・プログラミング入門
開催日時　：2月14日(金)
開催場所　：日本テクトロニクス(東京都品川区)
受講料　　：無料
問い合わせ先：日本テクトロニクススクール窓口，☎(03)3448-3015
http://www.tektronix.co.jp/News/School/main.html

先端光実装技術コース
開催日時　：2月17日(月)，2月27日(木)，2月28日(金)，3月13日(木)，3月14日(金)　(計5日間)
開催場所　：かながわサイエンスパーク西棟内研修室(神奈川県川崎市)，職業能力開発総合大学校東京校(東京都小平市)
受講料　　：79,000円(全日程)，20,000円(1日単位受講)
問い合わせ先：(財)神奈川先端科学技術アカデミー教育交流部教育研修課，☎(044)819-2033　http://home.ksp.or.jp/kast/

JSP/Servletプログラミング入門
開催日時　：2月18日(火)～2月19日(水)
開催場所　：DISパソコンスクール(東京都文京区)
受講料　　：98,000円
問い合わせ先：(株)エイチアイICP事業部，☎(03)3719-8155，FAX(03)3793-5109　http://icp.hicorp.co.jp/

ISS製品トレーニングコース RealSecure 7.x
開催日時　：2月19日(水)～2月21日(金)
開催場所　：日本システムハウス(東京都新宿区)
受講料　　：240,000円
問い合わせ先：日本システムハウス，sales2@nsh.co.jp，☎(03)3366-3101

C++プログラマのためのCOM入門
開催日時　：2月24日(月)～2月25日(火)
開催場所　：DISパソコンスクール(東京都文京区)
受講料　　：94,000円
問い合わせ先：(株)エイチアイICP事業部，☎(03)3719-8155，FAX(03)3793-5109　http://icp.hicorp.co.jp/

TCP/IPによるI/O制御の実際～Ethernetを利用した組み込み機器の設計
開催日時　：2月28日(金)～3月1日(土)
開催場所　：CQ出版セミナールーム
受講料　　：25,000円
問い合わせ先：エレクトロニクス・セミナー事務局，☎(03)5395-2125

日程はすべて予定です．問い合わせ先にご確認のうえ，お出かけください．

読者の広場

Interfaceへの声

2003年1月号特集「作りながら学ぶコンピュータシステム技術」に関して

▷今回の特集は興味がある反面，同種の仕事に直接かかわっていない限り試してみるのも一苦労だ．実際，ボードの価格を調べてみると，普通のパソコン部品と比べても(当然だが)かなり高価．FPGAボード1枚くらいなら何とかなるのですが．（信）

▷今回の特集はたいへん興味深く読みました．ただ，もっとも関心があったバックアップ付きリアルタイムクロックによるATX電源制御が誌面の都合で割愛されていたのが残念でした．次の機会に，ぜひとも詳しく取り上げてください．（玉出のタマ）

▷PCの高クロック化は目を見張るものがあります．組み込み用途にそれほどの性能はあまり必要ないですが，それでも大いに興味があります．数百MHzの信号が通る基板の実装はどんな設計なのでしょうか．最新技術の解説もお願いします．（はくりゅう）

▷SH-4＋PCIバスに関するハードウェア開発に携わっているので，特集はたいへん参考になりました．まもなくリリースされるSH7750Rは240MHz動作でキャッシュ容量が2倍になるということなので，楽しみです．（白石隆）

▷ソフトウェア一筋で生きてきたので，正直，今回の特集はよくわかりませんでしたが，オリジナルアーキテクチャのコンピュータを作るというのは，とてもロマンがあります．（金無し父さん貧乏父さん）

その他

▷「フジワラヒロタツの現場検証」は，非常に共感できると思っていましたが，じつは年齢も近い同世代の方だと今月号でわかりました．単行本化をめざして連載を続けてください．（JR9JUK）

▷別冊付録の「組み込みLinuxを使ったシステム設計の勘所」がとても役に立ちました．とくにTRONからLinuxへと移行する際のポイントがよく解説されていて，わかりやすかった．（ビギナーズ）

アンケートの結果

特集「作りながら学ぶコンピュータシステム技術」についてのアンケート

Q1　今回の特集のアプローチをどう思われましたか？

①非常におもしろくたいへん興味がある（44%）
②仕事上でも十分に参考になる（12%）
③学習/評価用に役に立つ（22%）
④話としてはおもしろい（22%）
⑤何の役にも立たない（0%）
⑥こんな企画はやめてほしい（0%）

Q2　もしあなたがCPU/メモリ/拡張バスを選定するならどれを選びますか?

▶CPU
①x86系（44%）　②MIPS系（0%）
③PowerPC系（34%）　④ARM系（0%）
⑤SH系（22%）

▶メモリ
①SDRAM（33%）　②DDR-SDRAM（33%）
③RDRAM（12%）　④EDO-DRAM（0%）
⑤SRAM（11%）　⑥その他（11%）

▶拡張バス
①PCI（78%）　②ISA（0%）　③VME（22%）
④Cバス（0%）　⑤オリジナルバス（0%）

Q3　設計/製作記事を掲載してほしいバスやインターフェース，または機能があればお書きください．

USB2.0，IEEE1394，無線LAN，光ファイバ

特集担当デスクから

☆Suicaの登場により，一気に身近になった感のあるICカードだが，ICカードに関する情報は意外と少なく，あったとしても「使う側」の情報がほとんどだった．そこで本特集では「作る側」のための情報――アプリケーション開発のために必要となる情報を集めてみた．
☆情報が少ないこともあり，なにやらブラックボックスのように思われるICカードだが，結局はCPUとメモリが搭載されてOSとアプリケーションが動作している，ごく普通の組み込み機器なのである．
☆あとはアプリケーションの充実である．まずは財布の中にある十数枚のポイントカードをICカードにまとめたい．そのほかにもチケット購入と本人確認システムを連携させ，入場口でいちいち年齢確認をする手間を省きたい．非実用的な用途としては，いわゆる電子ペット的なものも可能かもしれない……成長を確認するために端末が必要となるが．
☆しかし，なぜ人はカードに惹かれるのだろうか？　携帯性を考えると時計などと一体化したほうが合理的かもしれないし，実際にFeliCaの応用例では時計型のものも存在するという．実装の容易さなどを考えると，カード形状を選択するというのは非合理的なことかもしれない．それでも人はカード化することを選択する．なぜか．もしかしたら，それはカードのもつ魔力のためなのかもしれない．

次号予告 解説！USBの徹底活用技法

2003年4月号は2月25日発売です

EZ-USB FX2/OHCI/UHCI/On-The-Go/組み込み向けUSBプロトコルスタック/USB HUB

OSのドライバなどの問題から立ち上がりが遅かったUSB2.0も，最近ではUSB2.0対応周辺機器も増えてきた．最新のマザーボードではオンボードでUSB2.0インターフェースを搭載してくるなど，USB2.0も普及段階に入ったといえる．しかし，USB2.0対応のターゲット機器の設計事例はまだまだ少ない．そこで次号では，USB2.0対応USB学習キットをとりあげ，480Mbpsの高速転送を活かしたターゲットシステムの設計事例を解説する．

またUSBは，PCだけでなく組み込み機器でも使われ始めている．これまでのPC周辺機器としてのUSBターゲットデバイスではなく，それを制御するUSBホストを実現する要求も高まっている．そこで，組み込み機器にUSBホスト機能を実現するためのホストコントローラについて，また組み込み向けのUSBプロトコルスタック/ミドルウェアなどについても解説する．さらに，On-The-Go対応USBデバイス，そしてUSB2.0の帯域を活かすUSB HUBチップなどについても解説する．

★次号には，『移り気な情報工学』が別冊付録として付きます！

編集後記

■本欄を書いている今日はX'mas．朝，昨夜の仕事(？)でボケた耳に「やったぁ！プレゼントだ！サンタさんありがとう！」の声――信じているかわからないけれど，聞く側は嬉しい．こちらこそありがとう．姉(8才)がツリーの靴下にサンタさん宛てに入れたメッセージは「ほうせき」，妹(6才)のは「いるかのおもちゃ」．それぞれにGet．（洋）

■「災い転じて福となす」というが，これはかなりの長期的視野に立ってみないことには，災いが本当に災いでしかなかったのか，それとも福に転じたのかがわかるものではない．災いの最中には不安になりこそはすれ，希望を抱くことすらできないのが普通だろう．先行きの見えない世の中を見て，ふとこんなことを思った次第．（＝IO）

■某誌付録の某公式エミュレータをGetし，学生時代に作ったプログラムを走らせると…DOSアプリの類は当然として，VDP(ビデオコントローラ)直たたきプログラムや，DISK BIOSまわりをごにょごにょ(笑)するドライバもちゃんと動く互換性の高さに驚く．そしてもう一つ…押入から発掘した十年以上昔の2DDのFDがまだ読めたコト！（M）

■自分へのクリスマスプレゼントとして，販売終了になるキーボードを3台，慌てて駆け込み購入することになりました．すでに同じキーボードを3台持っているので都合6台になるのですが，これだけあれば一生もつでしょう．その前にPS/2インターフェースがなくなるような気もしますが……．（み）

■パソコンの次のキーワードは，同期化ソフトになるらしい．PDAではHotSyncが主流になっているが，パソコンではアップルはiSync，マイクロソフトはActiveSyncを用意している．使ってみれば便利なことがわかるが，最初は何に使うのかがわからなかった．さまざまなデータを，必要なときに簡単に取り出すということは意外と大変なのである．（Y）

■たばこの税金が上がるとか．ヘビーという程では無いにしても喫煙者の私には懐がいたむ話です．これを機に禁煙すると良いとも言われてちょっと考えましたが，軟弱ものの私にはそれも難しい話かと．体に悪いということは重々わかってはいるのですが……．とりあえずは，一日の本数を減らし遣り繰りしていくしかないのかな．（Y2）

■子供のクリスマスプレゼントを買いにデパートに行った．子供服とおもちゃ売り場はこの不景気どこ吹く風で大盛況．福沢諭吉が面白いように飛び交っていた．少子化が言われ続けて久しい．年金問題も含めて将来の担い手として語られるケースが多いが，需要を引っ張る子供の役割を目の当たりにするとより切実に痛感してしまう．（ちゃん）

■先日沖縄へ行った．「琉球料理を宿のオーナーと作る」オプションを楽しみにしていた．作ったものは4品で，全てチャンプルー．様子を見に来た私の連れに，「今，彼女がおいしいご飯作ってるから待っててねぇ」と言いながらCampbellスープの缶詰を鍋へ移していた．おかしいぞ．（米）

お知らせ

▶読者の広場
本誌に関するご意見・ご希望などを，綴じ込みのハガキでお寄せください．読者の広場への掲載分には粗品を進呈いたします．なお，掲載に際しては表現の一部を変更させていただくことがありますので，あらかじめご了承ください．

▶投稿歓迎
本誌に投稿をご希望の方は，連絡先(自宅/勤務先)を明記のうえ，テーマ，内容の概要をレポート用紙1～2枚にまとめて「Interface投稿係」までご送付ください．メールでお送りいただいても結構です(送り先はsupportinter@cqpub.co.jpまで)．追って採否をお知らせいたします．なお，採用分には小社規定の原稿料をお支払いいたします．

▶本誌掲載記事についてのご注意
本誌掲載記事には著作権があり，示されている技術には工業所有権が確立されている場合があります．したがって，個人で利用される場合以外は，所有者の許諾が必要です．また，掲載された回路，技術，プログラムなどを利用して生じたトラブルについては，小社ならびに著作権者は責任を負いかねますので，ご了承ください．
本誌掲載記事をCQ出版(株)の承諾なしに，書籍，雑誌，Webといった媒体の形態を問わず，転載，複写することを禁じます．

▶コピーサービスのご案内
本誌バックナンバーの掲載記事については，在庫(原則として24か月分)のないものに限りコピーサービスを行っています．コピー体裁は雑誌見開きの，複写機による白黒コピーです．なお，コピーの発送には多少時間がかかる場合があります．

- コピー料金(税込み)
  1ページにつき100円
- 発送手数料(判型に関わらず)
  1～10ページ：100円，11～30ページ：200円，31～50ページ：300円，51～100ページ：400円，101ページ以上：600円
- 送付金額の算出方法
  総ページ数×100円＋発送手数料
- 入金方法
  現金書留か郵便小為替による郵送
- 明記事項
  雑誌名，年月号，記事タイトル，開始ページ，総ページ数
- 宛て先
  〒170-8461　東京都豊島区巣鴨1-14-2
  CQ出版株式会社　コピーサービス係
  (TEL：03-5395-4211，FAX：03-5395-1642)

▶お問い合わせ先のご案内
- 在庫，バックナンバー，年間購読送付先変更に関して
  販売部：03-5395-2141
- 広告に関して
  広告部：03-5395-2133
- 雑誌本文に関して
  編集部：03-5395-2122

記事内容に関するご質問は，返信用封筒を同封して編集部宛てに郵送してくださるようお願いいたします．筆者に回送してお答えいたします．

Interface
©CQ出版(株)　2003　振替 00100-7-10665
2003年3月号　第29巻　第3号(通巻第309号)
2003年3月1日発行(毎月1日発行)
定価は裏表紙に表示してあります

発行人／蒲生良治
編集人／相原　洋
編集／大野典宏　村上真紀　山口光樹　小林由美子
デザイン・DTP／クニメディア株式会社
表紙デザイン／株式会社プランニング・ロケッツ
本文イラスト／森　祐子
広告／澤辺　彰　中元正夫　渡部真美

発行所／CQ出版株式会社　〒170-8461　東京都豊島区巣鴨1-14-2
電話／編集部 (03) 5395-2122　URL http://www.cqpub.co.jp/interface/
広告部 (03) 5395-2133　インターフェース編集部へのメール
販売部 (03) 5395-2141　supportinter@cqpub.co.jp
CQ Publishing Co.,Ltd.／1-14-2 Sugamo, Toshima-ku, Tokyo 170-8461, Japan
印刷／クニメディア株式会社　美和印刷株式会社
製本／星野製本株式会社

日本ABC協会加盟誌
(新聞雑誌部数公査機構)　ISSN0387-9569

Printed in Japan